ZX Spectrum +2

sinclair



Manual del Usuario



ZX Spectrum +2A
Manual del usuario

El producto descrito en este manual, así como los diseñados para ser utilizados con él, están sujetos a desarrollo y mejoras continuas. En particular, puede haber diferencias entre los mensajes que aparezcan en su pantalla y los que se muestra en este manual. Toda la información técnica relativa al producto y su utilización (incluida la que figura en este manual) es suministrada por Amstrad con la convicción de que es correcta.

Toda reparación u operación de mantenimiento de este producto debe ser confiada a los talleres autorizados por AMSTRAD ESPAÑA. Amstrad no puede asumir ninguna responsabilidad derivada del daño o pérdida que se pueda ocasionar como resultado de reparaciones efectuadas por personal no autorizado. El objetivo de este manual no es sino servir de ayuda al usuario en la utilización del producto; por consiguiente, Amstrad queda eximido de responsabilidad por el daño o pérdida a que pueda dar lugar la utilización de la información aquí publicada o la incorrecta utilización del producto.

Rogamos a los usuarios que rellenen y envíen las tarjetas de registro/garantía.

Amstrad agradecerá el envío de comentarios y sugerencias relativos a este manual y al producto en él descrito.

Toda la correspondencia se debe dirigir a:

AMSTRAD ESPAÑA
Aravaca 22
28040 Madrid

Publicado por Amstrad España
Escrito por Ivor Spital, con la colaboración de Cliff Lawson y Rupert Goodwins
Producido por Des Rackliff
Incluye extractos de la obra 'ZX Spectrum BASIC programming', de Steven Vickers y Robin Bradbeer
+3DOS ha sido desarrollado por Locomotive Software Ltd
Traducido por Emilio Benito Santos
Edición española producida por Vector Ediciones

Primera edición: 1987

IMPORTANTE

(No se preocupe si por ahora no entiende toda la jerga técnica utilizada en esta sección. El significado de estas advertencias se le irá haciendo evidente a medida que avance por el manual.)

Por favor, lea las siguientes advertencias:

1. No intente conectar este equipo a una red de distribución de energía eléctrica que no sea de 220-240 V c.a., 50 Hz.
2. Cuando haya terminado de usar el **+2A**, desconecte *siempre* la fuente de alimentación de la red.
3. El mantenimiento que pueda hacer el usuario no requiere en ningún caso acceder al interior del equipo. Así pues, NO ABRA NUNCA LA FUENTE DE ALIMENTACIÓN, porque en su interior hay ALTA TENSIÓN. Confíe todas las reparaciones a personal cualificado.
4. No obstruya ni cubra los orificios de ventilación.
5. No utilice ni almacene el equipo a temperaturas demasiado altas ni demasiado bajas, ni en lugares húmedos o polvorientos.
6. No conecte ni desconecte ningún dispositivo en los zócalos de la cara posterior del **+2A** cuando éste se encuentre encendido, pues corre el grave riesgo de dañar tanto el ordenador como el dispositivo externo.
7. Si tiene conectada una unidad de disco externa (con su interfaz), mantenga el cable de datos alejado de los cables de alimentación.
8. Cuando haya encendido el televisor (o el monitor), no apague el **+2A** inmediatamente; espere unos segundos.
9. No apague el ordenador cuando en la memoria del **+2A** tenga algún programa o datos que desee conservar, pues los perdería inmediatamente. Además, si enciende o apaga cualquier periférico puede provocar la «caída del sistema», lo que representa la pérdida del programa y los datos.

Si utiliza discos:

10. Evite que la unidad de disco y los propios discos estén sometidos a la influencia de campos magnéticos externos. Es decir, manténgalos alejados del televisor o monitor y de cualquier otra fuente de interferencia eléctrica.

11. No encienda ni apague el equipo mientras haya un disco en la unidad, so pena de perder la información en él grabada.

12. Haga copias de seguridad de todos los discos que contengan datos o programas valiosos, pues si se deteriorasen accidentalmente su sustitución podría ser muy costosa.

13. No toque la superficie sensible de los discos. No abra la caja de protección ni intente introducir ningún objeto en ella.

14. No extraiga el disco mientras el ordenador no haya terminado de leer o escribir en él.

15. Recuerde que al formatear los discos se borra toda la información que pudieran contener.

Contenido

Introducción

Sinclair ZX Spectrum +2A Ordenador doméstico de 128K

El Sinclair ZX Spectrum **+2A** es una versión ampliada del Spectrum **+2**. El **+2A** ha sido diseñado para incorporar gran parte de las características más avanzadas del Spectrum **+3**, entre las que se encuentran: el disco de RAM interno, con su sistema operativo de disco (que también puede gestionar las unidades de disquete externas), un interfaz paralelo (para conectar una impresora de tipo 'Centronics' estándar) y un interfaz auxiliar (para controlar otros dispositivos externos, por ejemplo, robots). Todo esto está integrado en un ordenador de 128K que incluye sonido, conectores para joysticks, interfaces MIDI y serie (RS232), junto con un completo 'bus' de ampliación y un magnetófono de cassette incorporado.

El conjunto es un sistema completo que sintetiza todo el ingenio de la tecnología Sinclair y la experiencia de Amstrad en materia de integración y fiabilidad. El **+2A** engloba, en una sola máquina, toda la potencia informática y toda la 'ampliabilidad' que usted puede necesitar.

Compatibilidad de los programas

El **+2A** puede funcionar con los programas escritos para modelos anteriores de la serie ZX Spectrum. Esto representa que ya hay una inmensa variedad de programas disponibles para el **+2A**; literalmente, miles de títulos que cubren todas las aplicaciones imaginables: juegos, utilidades, música, programas científicos y educativos, etc.

BASIC

El **+2A** utiliza un lenguaje de ordenador llamado *BASIC* (Beginners' All-purpose Symbolic Instruction Code: código de instrucciones simbólicas de uso general para principiantes). La versión de BASIC suministrada con el **+2A** es la misma del Spectrum **+3**. Se llama '+3 BASIC' y ha sido especialmente diseñada para facilitar su aprendizaje y utilización.

El sistema operativo de disco

El sistema operativo de disco del **+2A** es también el mismo que en el **+3**. Su nombre es '+3DOS'.

Gracias a +3DOS, usted puede conectar una o dos unidades de disquete externas, a través del interfaz adecuado, según se explica en el Capítulo 10.

El **+2A** incluye también un disco de RAM, controlado por +3DOS. El disco de RAM es la unidad de disco (volátil) interna del **+2A**, en la que se puede grabar y cargar programas a gran velocidad.

Si se conecta una unidad de disquete externa, el **+2A** se comporta a todos los efectos exactamente igual que el **+3**. (Hasta el menú de presentación mostrará '+3' como nombre del modelo.)

Cómo leer este libro

Para poder sacar el máximo partido de su **+2A** es esencial que lea ordenadamente toda la información proporcionada en este manual. Si se salta algún capítulo, es posible que luego no sea capaz de entender un capítulo posterior.

Así pues, le recomendamos que adopte el siguiente programa de lectura:

Capítulo 1. Este capítulo explica cómo se conecta el sistema **+2A**.

Capítulo 2. Este capítulo explica cómo se enciende el equipo y cómo se sintoniza el televisor para que muestre en la pantalla la señal procedente del ordenador. Después explica cómo se elige una opción en el menú de presentación. Todo esto es imprescindible para poder usar el **+2A**. No obstante, si usted ya sabe sintonizar el televisor y elegir opciones de menú (posiblemente porque ya ha usado un Spectrum 128, un **+2** o un **+3**), puede saltarse este capítulo.

Capítulo 3. Este capítulo explica cómo cargar programas comerciales (por ejemplo, juegos) grabados en disco. Si no ha conectado una unidad de disquete externa o no piensa comprar nunca ninguno de esos programas, puede saltarse este capítulo.

Capítulo 4. Este capítulo explica cómo cargar programas comerciales (por ejemplo, juegos) grabados en cinta. Sólo debe saltárselo en el caso de que no piense comprar nunca ninguno de esos programas.

Capítulo 5. Este capítulo explica el manejo de la unidad de disco externa. Puede saltárselo si no tiene intención de conectar una unidad de disco. Por otra parte, si ha conectado también la segunda unidad de disco externa, debe interpretar que siempre que hablemos de 'la unidad de disco' en términos generales, estaremos refiriéndonos a ambas unidades.

Capítulo 6. Este capítulo da una introducción a +3 BASIC. En particular, describe el editor y ciertos aspectos de +3 BASIC en los que éste se distingue del BASIC de otros ordenadores. Por lo tanto, aunque usted tenga experiencia en la programación en BASIC, no deje de leer este capítulo. (Para seguir sus ejemplos en la unidad de disquete externa necesitará un disco CF-2 vacío.) No obstante, si ha comprado el **+2A** solamente para ejecutar programas comerciales (por ejemplo, juegos) y no piensa usarlo para programar en BASIC, puede omitir la lectura de este capítulo.

Capítulo 7. Éste es el capítulo que usted puede ignorar con toda tranquilidad. Está dedicado a describir el modo '48 BASIC' (en el que el **+2A** funciona igual que los Spectrum 'antiguos', tanto en lo referente a edición como a programación). Este modo no es aconsejable más que para cargar programas de cinta escritos para el Spectrum 48. Usted no debe usarlo para programar en BASIC; de hecho, desde 48 BASIC no es posible servirse de las características más avanzadas del **+2A** (unidades de disco, memoria adicional, interfaces **RS232/MIDI/AUX** y disco de RAM).

Capítulo 8. Este capítulo forma el núcleo central del manual. Es una guía completa de programación en +3 BASIC. Si usted ya ha programado en otras versiones de BASIC, puede usar este capítulo como guía de referencia, consultando el índice para averiguar qué sección puede interesarle leer. Si, por el contrario, no tiene experiencia en BASIC, debe leer una a una las secciones de este capítulo, al menos las primeras. Después, cuando ya sepa introducir y ejecutar un programa y haya aprendido los fundamentos de BASIC, quizá pueda saltarse alguna sección si está impaciente por llegar a otras posteriores que le atraigan más. Ahora bien, si ha comprado el **+2A** solamente para ejecutar programas comerciales (por ejemplo, juegos) y no piensa usarlo para programas en BASIC, puede omitir la lectura de este capítulo.

Capítulo 9. Este capítulo explica la utilización del **+2A** como calculadora. No es imprescindible leerlo.

Capítulo 10. Este capítulo trata la conexión de periféricos al **+2A**. Como periféricos se puede usar una o dos unidades de disquete, una impresora, una unidad de disco adicional, un joystick, etc. Antes de conectar cualquier dispositivo al **+2A**, consulte este capítulo para averiguar cómo debe hacerlo. Si, en cambio, sólo va a usar el sistema estándar (es decir, el ordenador conectado a un televisor), no es necesario que lo lea.

Capítulo 6. Este capítulo da una introducción a +3 BASIC. En particular, describe el editor y ciertos aspectos de +3 BASIC en los que éste se distingue del BASIC de otros ordenadores. Por lo tanto, aunque usted tenga experiencia en la programación en BASIC, no deje de leer este capítulo. (Para seguir los ejemplos en la unidad de disquete externa necesitará un disco CF-2 vacío.) No obstante, si ha comprado el +2A solamente para ejecutar programas comerciales (por ejemplo, juegos) y no piensa usarlo para programar en BASIC, puede omitir la lectura de este capítulo.

Capítulo 7. Este es el capítulo que usted puede ignorar, con toda tranquilidad. Está dedicado a describir el modo 48 BASIC (en el que el +2A funciona igual que los Spectrum 'antiguos', tanto en lo referente a edición como a programación). Este modo no es aconsejable más que para cargar programas de cinta escritos para el Spectrum 48. Usted no debe usarlo para programar en BASIC; de hecho, desde 48 BASIC no es posible servirse de las características más avanzadas del +2A (unidades de disco, memoria adicional, interfaces RS232, MIDI/AUX y disco de RAM).

Capítulo 8. Este capítulo forma el núcleo central del manual. Es una guía completa de programación en +3 BASIC. Si usted ya ha programado con otras versiones de BASIC, puede usar este capítulo como guía de referencia, consultando el índice para averiguar qué sección puede interesarle leer. Si, por el contrario, no tiene experiencia en BASIC, debe leer todas las secciones de este capítulo (al menos las primeras). Después, cuando ya sepa introducir y ejecutar un programa y haya entendido los fundamentos de BASIC, quizá pueda saltarse alguna sección si está impaciente por llegar a otras posteriores que le atraigan más. Ahora bien, si ha comprado el +2A solamente para ejecutar programas comerciales (por ejemplo, juegos) y no piensa usarlo para programar en BASIC, puede omitir la lectura de este capítulo.

Capítulo 9. Este capítulo explica la utilización del +2A como calculadora [illegible]

Capítulo 10. Este capítulo trata la conexión de periféricos al +2A. Como periféricos se puede usar una o dos unidades de disquete, una impresora, una unidad de disco adicional, un joystick, etc. Antes de conectar cualquier dispositivo al +2A, consulte este capítulo para averiguar cómo debe hacerlo. [illegible]

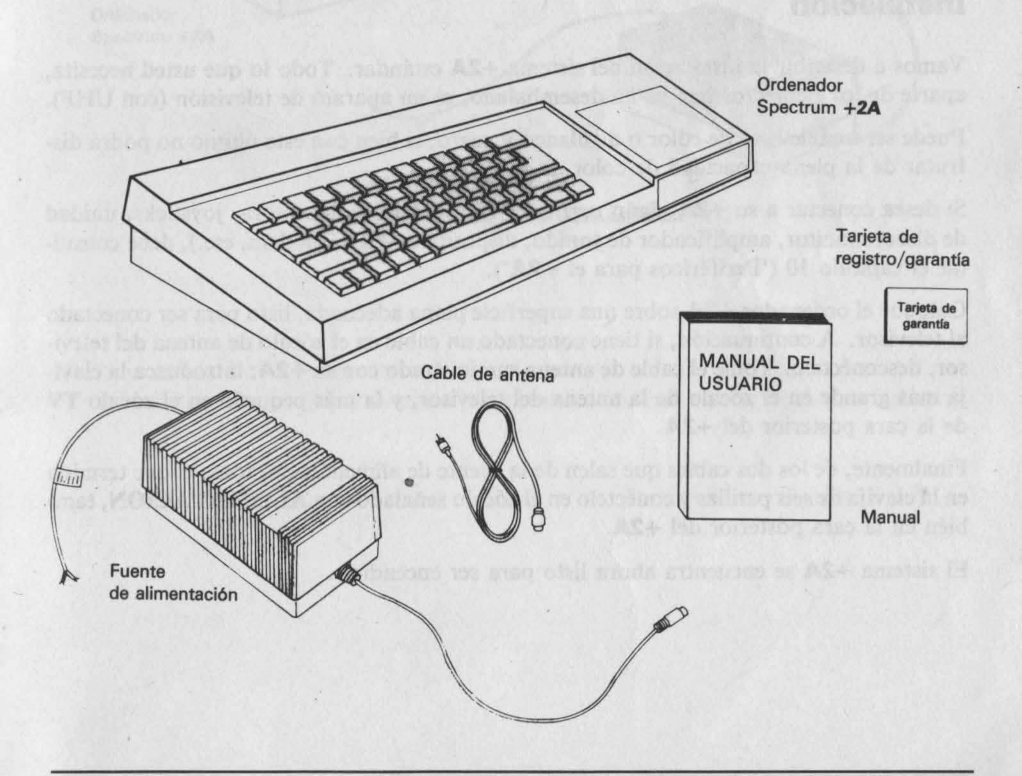
Apertura de la caja

Temas tratados en este capítulo:

Desembalaje
Conexión del ordenador a la red
Instalación

Desembalaje

La caja en que se suministra este ordenador debe contener lo siguiente:



- El ordenador Spectrum **+2A**
- La unidad de alimentación
- El cable de antena
- Este manual (junto con las tarjetas de registro y garantía)

Conexión del ordenador a la red

El Spectrum **+2A** sólo puede ser conectado a la red de 220-240V c.a., 50Hz.

No extraiga ningún tornillo ni trate de abrir la carcasa de la unidad de alimentación. Respete las advertencias impresas en la etiqueta de características, que está situada en la cara inferior de la fuente de alimentación:

> Desconecte esta unidad de la red cuando no la esté usando.
>
> ¡ATENCIÓN! No extraiga ningún tornillo. Circuitos activos en el interior.

Instalación

Vamos a describir la instalación del sistema **+2A** estándar. Todo lo que usted necesita, aparte de los elementos que ya ha desembalado, es un aparato de televisión (con UHF).

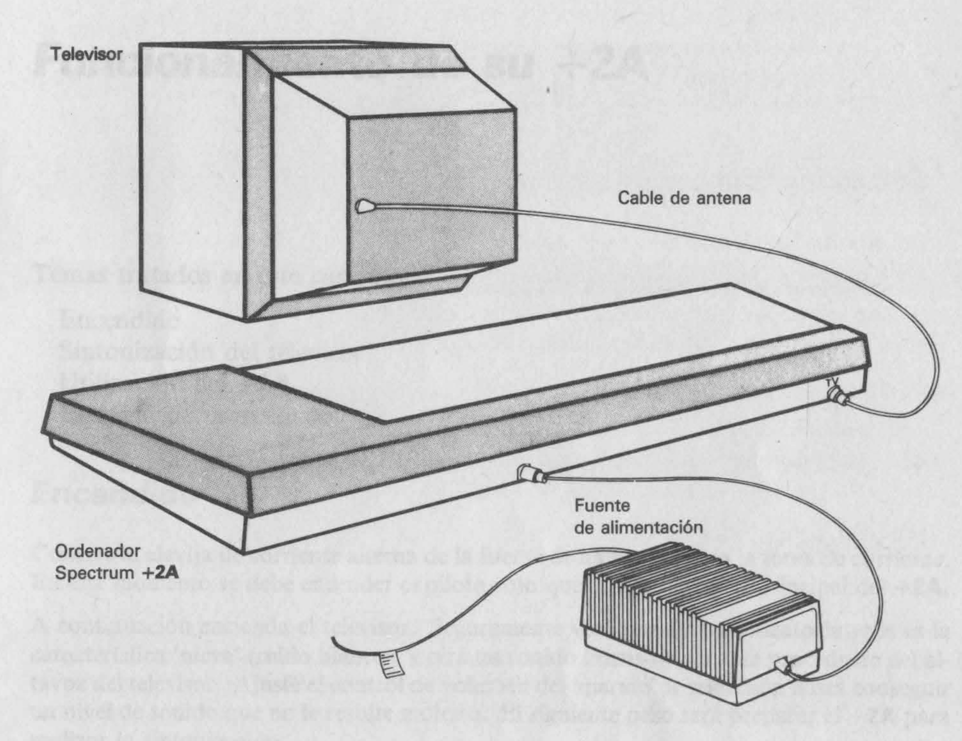
Puede ser un televisor de color o de blanco y negro, si bien con este último no podrá disfrutar de la plena capacidad de color de su **+2A**.

Si desea conectar a su **+2A** algún *periférico* (por ejemplo, impresora, joysticks, unidad de disco, monitor, amplificador de sonido, dispositivo MIDI, modem, etc.), debe consultar el capítulo 10 ('Periféricos para el **+2A**').

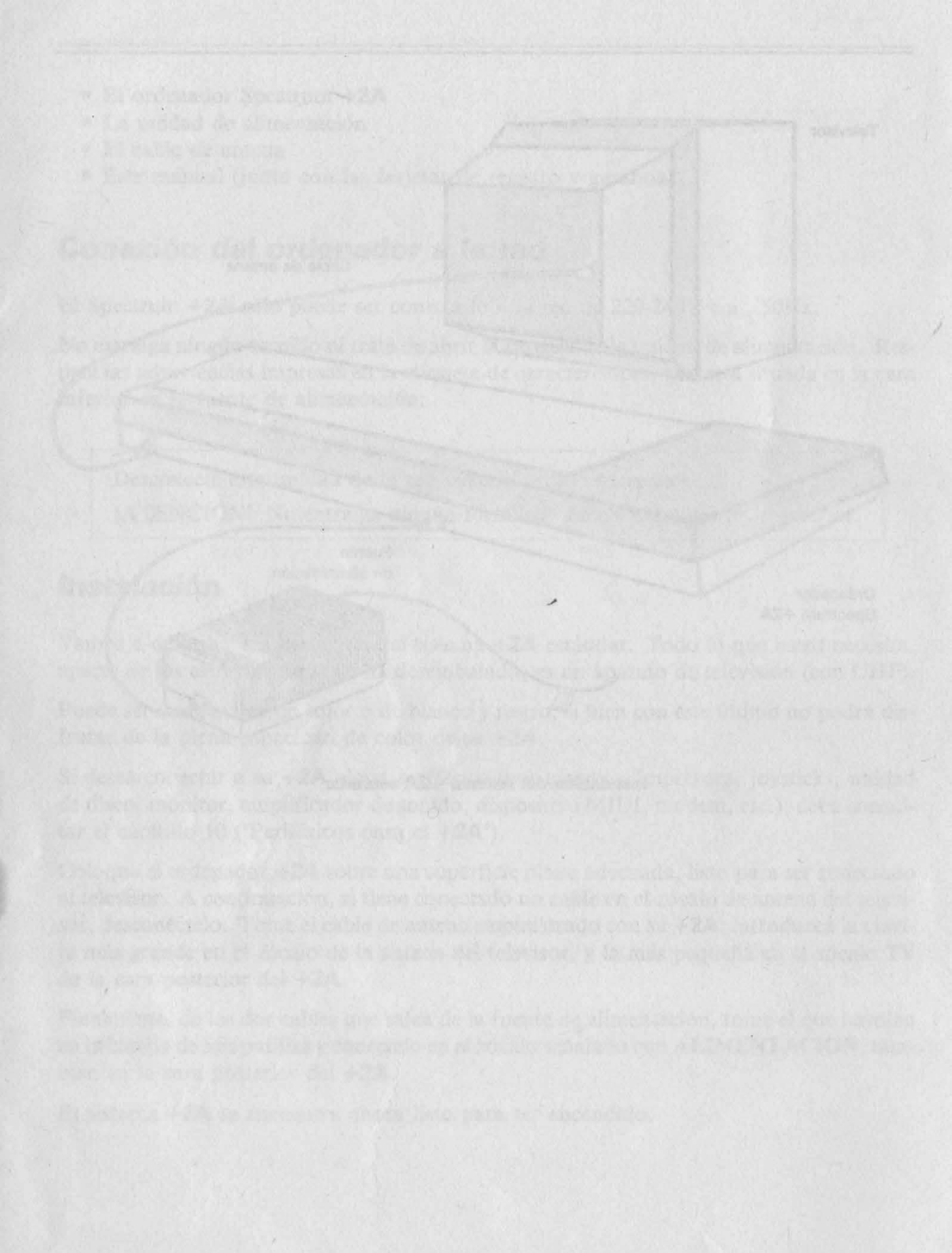
Coloque el ordenador **+2A** sobre una superficie plana adecuada, listo para ser conectado al televisor. A continuación, si tiene conectado un cable en el zócalo de antena del televisor, desconéctelo. Tome el cable de antena suministrado con su **+2A**; introduzca la clavija más grande en el zócalo de la antena del televisor, y la más pequeña en el zócalo **TV** de la cara posterior del **+2A**.

Finalmente, de los dos cables que salen de la fuente de alimentación, tome el que termina en la clavija de seis patillas y conéctelo en el zócalo señalado con **ALIMENTACION**, también en la cara posterior del **+2A**.

El sistema **+2A** se encuentra ahora listo para ser encendido.



Instalación del sistema +2A estándar



* El ordenador Spectrum +2A
* La unidad de alimentación
* El cable de antena
* Este manual (junto con las [illegible])

Conexión del ordenador a la red

El Spectrum +2A sólo puede ser conectado a la red de 220-240 V ca, 50Hz.

[illegible]

[illegible]

[illegible]

Instalación

[illegible] Todo lo que [illegible] un aparato de televisión (con UHF).

[illegible] color [illegible] +2A.

Si desea conectar a su +2A [illegible] joystick, unidad de disco, monitor, amplificador de sonido, dispositivo MIDI, módem, etc.), debe consultar el capítulo 10 ("Periféricos para el +2A").

Coloque el ordenador +2A sobre una superficie plana adecuada, listo para ser conectado al televisor. A continuación, si tiene conectado un cable en el zócalo de antena del televisor, desconéctelo. Tome el cable de antena suministrado con su +2A; introduzca la clavija más grande en el zócalo de la antena del televisor, y la más pequeña en el zócalo TV de la cara posterior del +2A.

Finalmente, de los dos cables que salen de la fuente de alimentación, tome el que termina en [illegible] conectado en el zócalo señalado con ALIMENTACIÓN, situado en la cara posterior del +2A.

El sistema +2A se encuentra ahora listo para ser encendido.

Funcionamiento de su +2A

Temas tratados en este capítulo:

Encendido
Sintonización del televisor
Utilización del **+2A**
El menú de presentación

Encendido

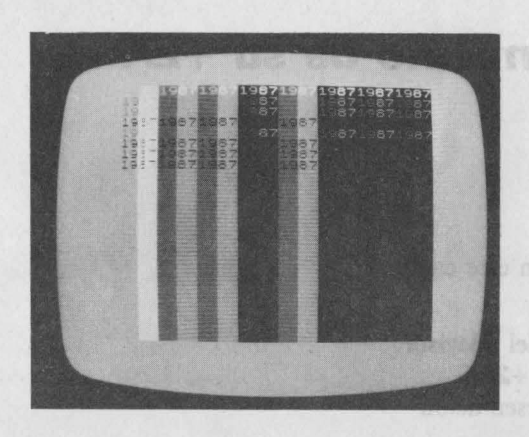
Conecte la clavija de corriente alterna de la fuente de alimentación a la toma de corriente. En este momento se debe encender el piloto rojo que está en el panel principal del **+2A**.

A continuación encienda el televisor. Seguramente verá una imagen aleatoria, que es la característica 'nieve' (ruido blanco), y oirá un sonido intenso y silbante procedente del altavoz del televisor. Ajuste el control de volumen del aparato de televisión hasta conseguir un nivel de sonido que no le resulte molesto. El siguiente paso será preparar el **+2A** para realizar la sintonización.

Preparativos para sintonizar el televisor

El **+2A** es capaz de generar su propia 'carta de ajuste', mediante la cual podemos sintonizar el televisor con toda precisión. Esa carta de ajuste consiste en dieciséis barras verticales de color (con caracteres de texto sobreimpresos) que aparecen en la pantalla del televisor y un sonido repetitivo que es reproducido por el altavoz. (Si el televisor es de blanco y negro, las barras de color aparecerán en diversos matices de gris). Podrá oír y ver esta señal de comprobación cuando haya terminado de sintonizar el televisor (por el procedimiento descrito en la sección siguiente).

Para activar la señal de prueba, pulse la tecla BREAK (que se encuentra en el extremo superior derecho del teclado) y, antes de soltarla, pulse y suelte el botón **REINIC** (que está en la cara izquierda del **+2A**). Mantenga pulsada la tecla BREAK durante unos segundos más y luego suéltela. El **+2A** empezará a generar la 'carta de ajuste' y usted podrá comenzar a sintonizar el televisor, tal y como se explica a continuación.



Sintonización cuando el televisor tiene selector de canales de botonera

Si su aparato de televisión no tiene selector de canales de botonera, pase a la sección siguiente.

Si el selector de canales sí es de ese tipo, pulse uno de los botones para seleccionar un canal que esté libre (es decir, uno que no esté utilizando para recibir programas de televisión o de video). Si el televisor dispone de 'control automático de frecuencia' (AFC o AFT), debe desconectarlo.

Ajuste el mando de sintonía correspondiente al canal elegido hasta que la pantalla muestre la imagen de la figura anterior y el sonido sea lo más limpio posible.

Una vez conseguida la sintonía, ya puede conectar el control automático de frecuencia del televisor.

Finalmente, ajuste los mandos de contraste, color y brillo del televisor para optimizar la legibilidad de los caracteres de texto.

Ahora que ha sintonizado un canal específicamente para el **+2A**, cada vez que desee utilizar el ordenador junto con el televisor bastará con que seleccione ese canal.

Si todo ha ido bien, puede pasar directamente a la sección titulada 'Utilización del **+2A**'. Si no, consulte la sección '¿Algún problema?'.

Sintonización manual

Si su televisor no está equipado con selector de canales de botonera, tendrá que utilizar el mando de sintonía manual para ajustarlo al **+2A**.

Después de conectar y encender el **+2A** y el televisor, active la señal de prueba según se ha explicado en la sección 'Preparativos para sintonizar el televisor'.

Ajuste el mando de sintonía hasta que la pantalla muestre la carta de ajuste y el sonido sea lo más limpio posible.

Finalmente, ajuste los mandos de contraste, color y brillo del televisor para optimizar la legibilidad de los caracteres de texto.

Cada vez que desee utilizar el **+2A** junto con el televisor, deberá seguir este procedimiento de sintonización manual.

Si todo ha ido bien, puede pasar directamente a la sección titulada 'Utilización del **+2A**'. Si no, consulte la sección siguiente.

¿Algún problema?

Si ha conseguido sintonizar el televisor satisfactoriamente, puede pasar a la siguiente sección.

Si no es capaz de sintonizarlo, la siguiente lista de comprobación puede ayudarle a determinar dónde reside el problema y qué remedio aplicarle.

1. Problema: **No se enciende el piloto rojo de alimentación (situado en el panel superior del +2A).**

   Remedios:
   - Compruebe que la clavija de 6 patillas de la fuente de alimentación está conectada al zócalo **ALIMENTACION** del ordenador.
   - Compruebe que la clavija de corriente alterna de la fuente de alimentación se encuentra conectada a la toma de corriente.
   - Compruebe las conexiones dentro de la toma de corriente.

2. Problema: **El piloto de alimentación se enciende, pero no es posible sintonizar ninguna señal en el televisor.**

   Remedios:
   - Compruebe que el televisor está conectado y que funciona correctamente.
   - Compruebe que el televisor es del tipo UHF estándar (para color o para blanco y negro).

- Compruebe que el cable de antena (suministrado con el ordenador) está bien conectado al zócalo de antena del televisor y al del ordenador.
- Si el televisor tiene selector de canales por botonera, compruebe que está pulsado el botón correspondiente al canal elegido.

3. Problema: **La señal que se consigue en el televisor, procedente del ordenador, es de baja calidad.**

Remedios:
- Compruebe que el aparato de televisión está conectado y que funciona correctamente.
- Compruebe que el cable de antena (suministrado con el ordenador) está bien conectado al zócalo de antena del televisor y al del ordenador.
- Si el televisor dispone de control automático de frecuencia (AFC), desactívelo.
- Asegúrese de que ha realizado la sintonización lo más cuidadosamente posible.

4. Problema: **Se ha conseguido sintonizar una señal procedente del ordenador, pero no se trata de la carta de ajuste descrita anteriormente.**

Remedio:
- Asegúrese de que el ordenador está enviando al televisor la señal de prueba; consulte la sección titulada 'Preparativos para sintonizar el televisor'.

5. Problema: **Aparecen las barras de color de la carta de ajuste, pero no se oye ningún sonido (tono repetitivo) procedente de los altavoces.**

Remedios:
- Compruebe que el control de volumen del televisor no está al mínimo.
- Asegúrese de que ha realizado la sintonización lo más cuidadosamente posible.

6. Problema: **Se puede oír el sonido (tono repetitivo) de la señal de prueba, pero no se ve las barras de color en la pantalla.**

Remedios:
- Compruebe que los mandos de contraste, color y brillo del televisor no están al mínimo.
- Asegúrese de que ha realizado la sintonización lo más cuidadosamente posible.

7. Problema: **Se ha conseguido sintonizar las barras de color y el sonido de la señal de prueba, pero no es posible leer los caracteres de texto.**

Remedios:
- Asegúrese de que ha realizado la sintonización lo más cuidadosamente posible.
- Compruebe que los mandos de contraste, color y brillo del televisor han sido ajustados adecuadamente.

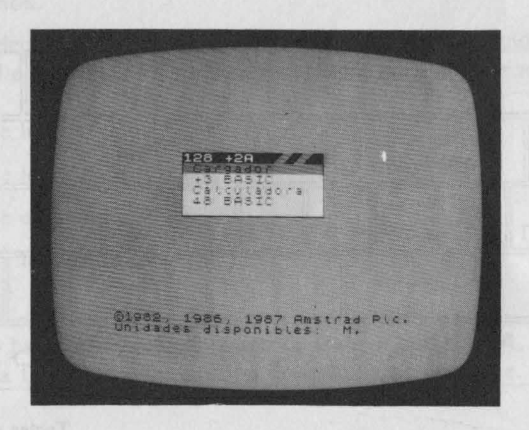
Si no logra identificar la causa de su problema, desarrolle de nuevo el procedimiento completo (desde el principio de este capítulo). Si el problema aún persiste, avise a su distribuidor.

Utilización del +2A

El sistema **+2A** debería estar ya completamente preparado, con las barras de color de la señal de prueba en la pantalla y el sonido repetitivo en los altavoces del televisor.

Ahora vamos a desconectar la señal de prueba y comenzaremos a utilizar el **+2A**. Pulse y suelte el botón **REINIC** (cara izquierda del **+2A**). Desaparecerá de la pantalla la carta de ajuste y en su lugar podrá ver el 'menú de presentación'.

El menú de presentación



El menú de presentación aparece cada vez que se enciende el **+2A**, y también cada vez que se lo reinicializa (pulsando y soltando el botón **REINIC**).

Si tiene conectada una unidad de disco externa (o las dos), el **+2A** se comporta a todos los efectos como si fuera un **+3**, hasta tal punto que en la primera línea del menú muestra '+3' como nombre del modelo.

Las opciones del menú de presentación son las cuatro que puede ver en un recuadro en la pantalla:

Cargador Elija esta opción si desea cargar programas escritos para el Spectrum 128, el Spectrum **+2**, el Spectrum **+2A** o el Spectrum **+3**.

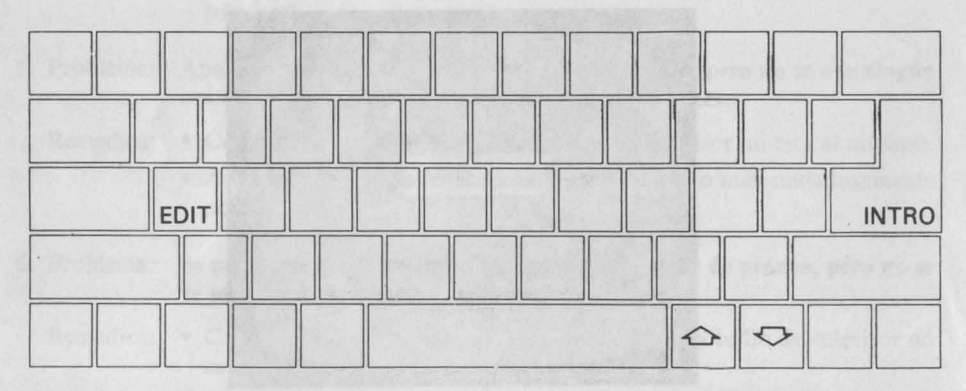
+3 BASIC Elija esta opción si desea utilizar el **+2A** para programar en BASIC.

Calculadora Elija esta opción si desea utilizar el **+2A** solamente como calculadora.

48 BASIC Elija esta opción si desea cargar (desde una unidad de cinta externa) programas escritos para el Spectrum 48, o si quiere utilizar el **+2A** como si fuese un Spectrum de 48K.

Cómo elegir una opción

Observe que la opción **Cargador** aparece resaltada por una 'barra'. Esto significa que dicha opción está preseleccionada, o sea, lista para ser seleccionada (la selección aún no ha sido confirmada). A los efectos de este ejemplo, supongamos que no queremos seleccionar **Cargador**, sino **+3 BASIC**. Esto implica que tenemos que llevar la barra a la línea de la opción **+3 BASIC**. Para ello, pulse las teclas del *cursor* (ilustradas en la figura siguiente) hasta que dicha barra llegue a la posición deseada.



Teclas del cursor

Cuando la barra esté sobre **+3 BASIC**, confirme la elección pulsando la tecla INTRO. El **+2A** activará entonces el modo +3 BASIC. (Verá una barra negra horizontal en la parte inferior de la pantalla y un cursor que parpadea en la esquina superior izquierda.)

No se preocupe si no sabe nada de BASIC, pues todavía no vamos a empezar a programar. Ahora vamos a volver al menú de presentación. Para ello utilizaremos un menú diferente, denominado *menú de edición*. Este menú se invoca pulsando la tecla EDIT.



Usando de nuevo las teclas del cursor e INTRO, seleccione la opción **Salida** para volver al menú de presentación.

Ahora puede usted seleccionar cualquier opción del menú de presentación. Dependiendo de su elección, consulte alguno de los siguientes capítulos para obtener más información:

Cargador Consulte los capítulos 3 y 4.

+3 BASIC Consulte los capítulos 6 y 8.

Calculadora Consulte el capítulo 9.

48 BASIC Consulte los capítulos 4 y 7.

Importante. Cuando haya terminado su sesión de trabajo con el **+2A, no olvide** desconectar la fuente de alimentación de la toma de corriente.

No se preocupe [illegible] de BASIC[illegible]. Ahora vamos a volver al menú de presentación. Para ello utilizaremos un menú alternativo denominado *menú de edición*. Este menú se invoca pulsando la tecla [EDIT].

Usando de nuevo las teclas del cursor y [ENTER], seleccione la opción Salida para volver al menú de presentación.

Ahora puede usted seleccionar cualquier opción del menú de presentación. Dependiendo de su elección, consulte alguno de los siguientes capítulos para obtener más información:

Cargador: Consulte los capítulos [illegible].

+3 BASIC: Consulte los capítulos [illegible].

Calculadora: Consulte el capítulo 6.

48 BASIC: Consulte los capítulos 4 y 5.

Importante: Cuando haya terminado su sesión de trabajo con el +2A, no olvide desconectar la fuente de alimentación de la toma de corriente.

Carga de programas de disco

Temas tratados en este capítulo:

Discos y unidades para el **+2A**
Carga de los programas
Interrupción del proceso de carga

Si no piensa usted conectar una unidad de disco externa al **+2A**, puede omitir la lectura de este capítulo (en el que vamos a estudiar el procedimiento de carga de los programas que usted puede adquirir grabados en disco).

(Los procedimientos de carga, grabación, formateado, etc. relacionados con la programación en BASIC serán descritos en el Capítulo 6 y en la Sección 20 del Capítulo 8.)

Discos y unidades para el +2A

Si desea conectar una unidad de disquete externa al **+2A**, debe adquirir el modelo AMSTRAD FD-1, junto con un interfaz adecuado (AMSTRAD SI-1). Consulte a este respecto el Capítulo 10, 'Periféricos para el **+2A**'.

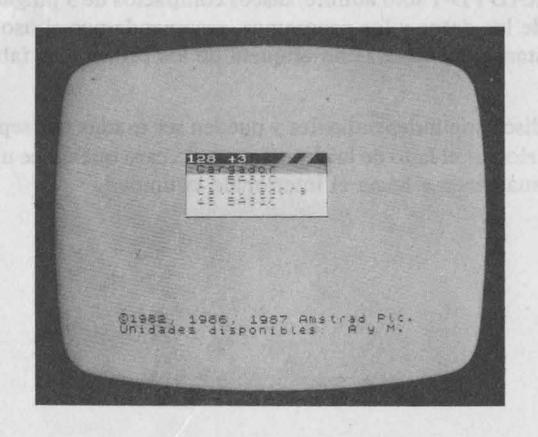
La unidad AMSTRAD FD-1 sólo admite 'discos compactos de 3 pulgadas'. Para garantizar la integridad de los datos y los programas, recomendamos el uso de discos CF-2 de AMSOFT; no obstante, los discos **con etiqueta** de los principales fabricantes son igualmente adecuados.

Las dos caras del disco son independientes y pueden ser usadas por separado. Para introducir el disco sujételo por el lado de la etiqueta, con la cara que desee utilizar hacia arriba, y hágalo deslizar suavemente hacia el interior de la unidad.

Carga de los programas

Para cargar programas comerciales (un juego, un programa educativo, etc.) grabados en disco, siga estas instrucciones:

1. Instale el **+2A** y la unidad FD-1. Encienda primero la FD-1 (con el interruptor situado en su panel posterior) y después el **+2A**. En la pantalla aparecerá el menú de presentación:



Puesto que tiene conectada la unidad de disco externa, el **+2A** se comporta exactamente igual que si fuera un **+3**, hasta el punto de que el menú de presentación muestra '+3' como nombre del modelo.

2. Introduzca en la unidad el disco del programa, con la cara que lo contiene hacia arriba.
3. Pulse INTRO para seleccionar la opción **Cargador** en el menú de presentación.

El **+2A** se pondrá a leer y cargar el programa. El piloto de la unidad de disco se encenderá intermitentemente (lo que indica que el ordenador está leyendo el disco). Al cabo de unos segundos el contenido de la pantalla cambiará y el programa habrá quedado listo para ser puesto en marcha.

Cuando haya terminado con el programa y quiera usar el ordenador para cualquier otra cosa, pulse y suelte el botón **REINIC** (que está en el lateral izquierdo del **+2A**). Recuerde siempre que cada vez que se pulsa el botón **REINIC** se borra todo lo que hay en la memoria (RAM) del ordenador. Por esa razón, **antes** de pulsar ese botón es necesario asegurarse de que no hay nada en la memoria del **+2A** cuya pérdida sea importante.

Si va a apagar el equipo, recuerde que antes debe extraer el disco de la unidad.

Interrupción del proceso de carga

Si desea abandonar la carga, pulse y suelte el botón **REINIC**. El **+2A** volverá a mostrar el menú de presentación.

Carga de los programas

Carga de programas de cinta

Temas tratados en este capítulo:

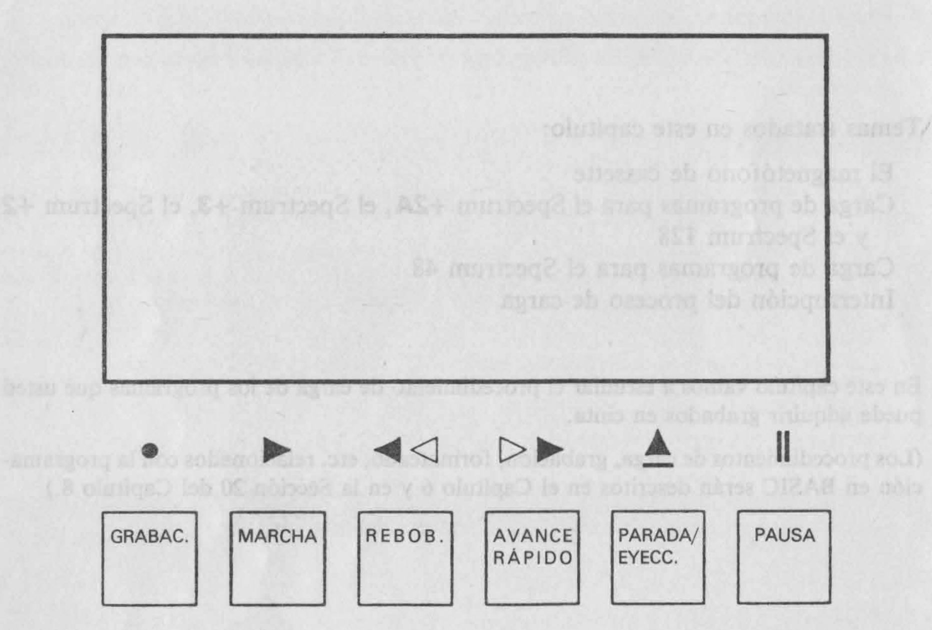
El magnetófono de cassette
Carga de programas para el Spectrum **+2A**, el Spectrum **+3**, el Spectrum **+2** y el Spectrum 128
Carga de programas para el Spectrum 48
Interrupción del proceso de carga

En este capítulo vamos a estudiar el procedimiento de carga de los programas que usted puede adquirir grabados en cinta.

(Los procedimientos de carga, grabación, formateado, etc. relacionados con la programación en BASIC serán descritos en el Capítulo 6 y en la Sección 20 del Capítulo 8.)

El magnetófono de cassette

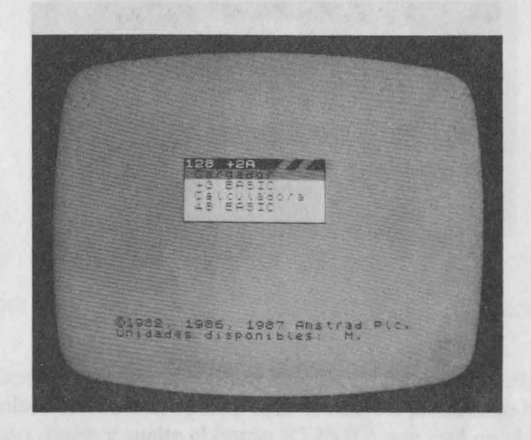
Antes de nada, observe los seis botones de control del magnetófono:



Programas para el Spectrum +2A, el Spectrum +3, el Spectrum +2 y el Spectrum 128

Para cargar programas comerciales (un juego, un programa educativo, etc.) diseñados para estos ordenadores y grabados en cinta, siga estas instrucciones:

1. Encienda o reinicialice el ordenador, de forma que la pantalla muestre el menú de presentación:



2. Si tiene conectada una unidad de disquete externa, compruebe que no hay ningún disco en ella.

3. Pulse INTRO para seleccionar la opción **Cargador** en el menú de presentación. (Si no sabe cómo seleccionar una opción del menú, consulte el Capítulo 2.)

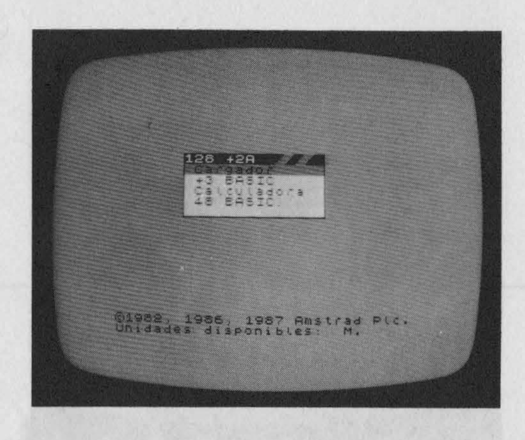
(Aunque tengamos conectada la unidad de disco, el **+2A** supone que queremos cargar un programa de cinta porque no hemos introducido ningún disco en la unidad. Si tuviéramos un disco en la unidad, el ordenador ignoraría la cinta.)

Ahora pase a la sección 'Carga desde la cinta'.

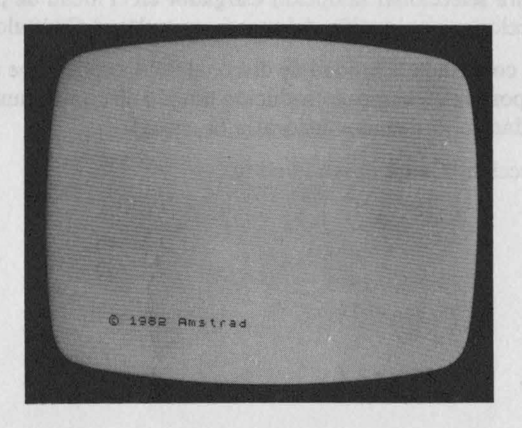
Programas para el Spectrum 48

Para cargar programas comerciales (un juego, un programa educativo, etc.) diseñados para el Spectrum 48, siga estas instrucciones:

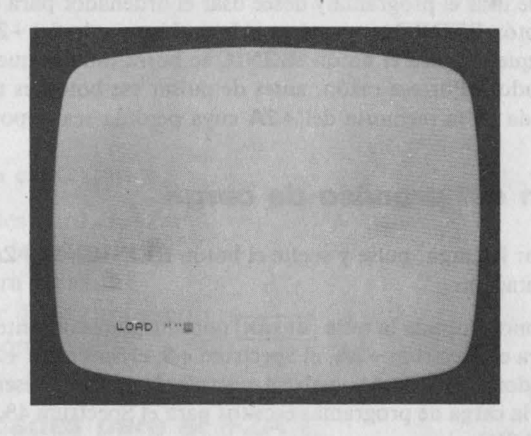
1. Encienda o reinicialice el ordenador, de forma que la pantalla muestre el menú de presentación:



2. Seleccione la opción **48 BASIC** en el menú de presentación. (Si no sabe cómo seleccionar una opción del menú, consulte el capítulo 2.) El menú de presentación desaparece y en la parte inferior de la pantalla se muestra el mensaje '**©1982 Amstrad**':



3. Ahora pulse una vez la tecla [J] y dos veces la tecla ["] (comillas). El aspecto de la pantalla debe ser el siguiente:



Cuando vea este mensaje pulse [INTRO]. Ahora pase a la sección 'Carga desde la cinta'.

(Si la imagen que aparece en la pantalla no es igual a la que se muestra en la figura anterior, quizás haya seleccionado una opción equivocada del menú o no haya pulsado la tecla correcta. En tal caso, pulse y suelte el botón **REINIC**, que está en el lateral izquierdo del **+2A**, y repita las etapas 2 y 3.)

Carga desde la cinta

1. Introduzca la cinta del programa en el magnetófono y asegúrese de que está rebobinada hasta el principio.
2. Ponga la cinta en movimiento. Cuando comience la carga, el color del margen parpadeará y aparecerá rayado; esto indica que el programa está siendo leído en la cinta. Si el mando de volumen del televisor no está al mínimo, oirá también un sonido variable de alta frecuencia. Es otra indicación de que el ordenador está cargando el programa.

La mayor parte de las cintas de programas disponibles comercialmente tarda unos pocos minutos en cargar. Inicialmente aparecerá el nombre del programa (**Programa:** *nombre*) en el extremo superior izquierdo de la pantalla, seguido de otros diversos mensajes e imágenes (que diferirán de un programa a otro.)

Cuando el programa esté cargado, pulse el botón de parada del magnetófono. El programa habrá quedado listo para ser puesto en marcha.

Cuando termine de usar el programa y desee usar el ordenador para cualquier otra cosa, pulse y suelte el botón **REINIC** (que está en el lateral izquierdo del **+2A**). Recuerde siempre que cada vez que se pulsa el botón **REINIC** se borra todo lo que hay en la memoria (RAM) del ordenador. Por esa razón, **antes** de pulsar ese botón es necesario asegurarse de que no hay nada en la memoria del **+2A** cuya pérdida sea importante.

Interrupción del proceso de carga

Si desea abandonar la carga, pulse y suelte el botón **REINIC**. El **+2A** volverá a mostrar el menú de presentación.

Nota. Si mantenemos pulsada la tecla [BREAK] unos instantes durante la carga de programas diseñados para el Spectrum **+2A**, el Spectrum **+3**, el Spectrum **+2** o el Spectrum 128, el ordenador abandona el proceso y vuelve a mostrar el menú de presentación. Si hacemos lo mismo durante la carga de programas escritos para el Spectrum 48, el ordenador retorna al modo 48 BASIC.

Unidades de disco

Temas tratados en este capítulo:

Discos y unidades para el **+2A**
Introducción de los discos
Protección contra escritura
Piloto indicador de lectura/escritura
Botón de eyección

Discos y unidades para el +2A

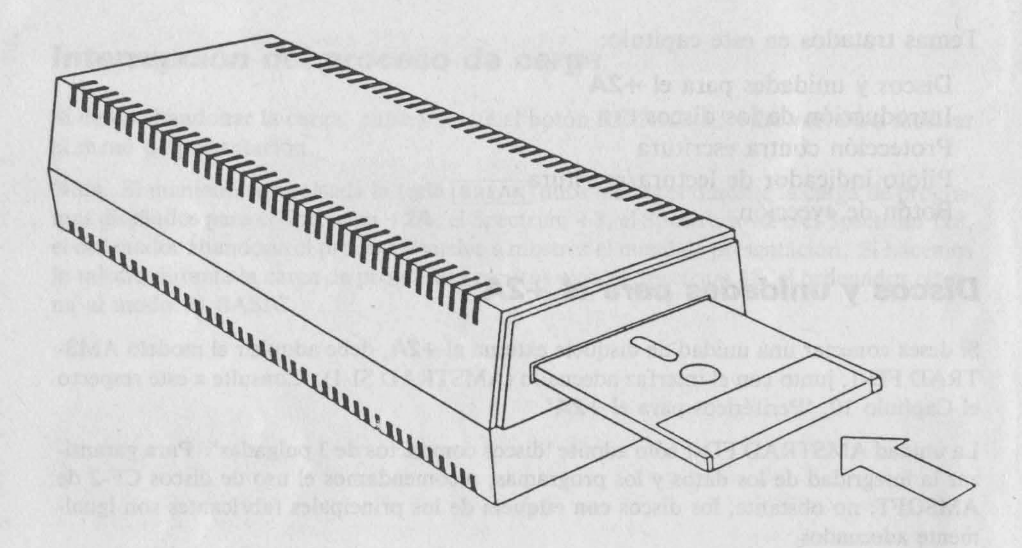
Si desea conectar una unidad de disquete externa al **+2A**, debe adquirir el modelo AMSTRAD FD-1, junto con el interfaz adecuado (AMSTRAD SI-1). Consulte a este respecto el Capítulo 10, 'Periféricos para el **+2A**'.

La unidad AMSTRAD FD-1 sólo admite 'discos compactos de 3 pulgadas'. Para garantizar la integridad de los datos y los programas, recomendamos el uso de discos CF-2 de AMSOFT; no obstante, los discos **con etiqueta** de los principales fabricantes son igualmente adecuados.

La primera unidad externa es la **unidad A**; la segunda unidad externa, si está conectada, es la **unidad B**.

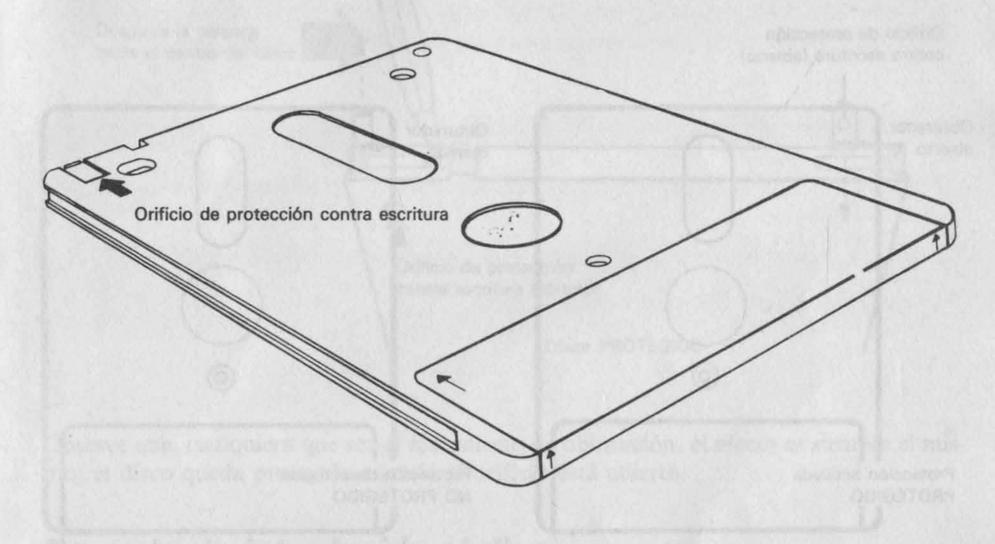
Introducción de los discos

Las dos caras del disco son independientes y pueden ser usadas por separado. Para introducir el disco sujételo por el lado de la etiqueta, con la cara que desee utilizar hacia arriba, y hágalo deslizar suavemente hacia el interior de la unidad.



Protección contra escritura

Una característica de los discos de 3 pulgadas es su sistema de protección contra escritura. Para cada cara hay un orificio de protección, situado en el extremo posterior izquierdo.



Cuando el orificio está **cerrado**, el disco está **desprotegido**; esto quiere decir que el ordenador puede 'escribir' en él. Si el orificio está **abierto**, el disco está **protegido**; es decir, el ordenador no puede escribir en el disco, y esto evita el borrado accidental de los programas y los datos.

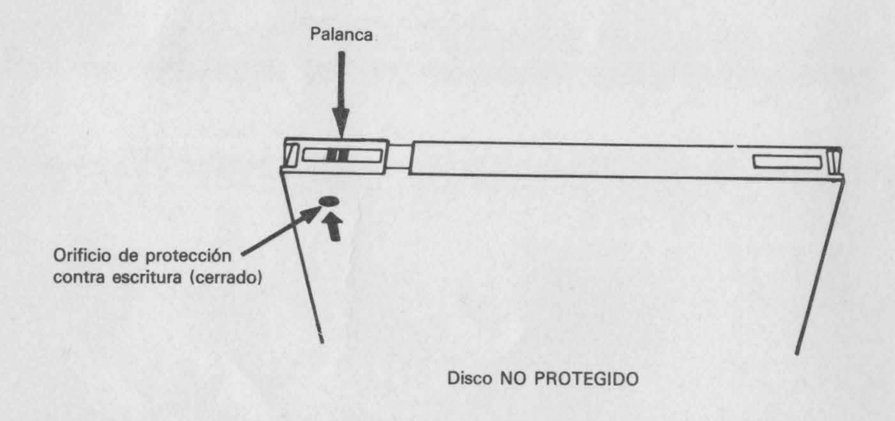
La forma de abrir y cerrar los orificios de protección depende de la marca del disco. En los discos AMSOFT CF-2 el funcionamiento es como sigue:

Para abrir el orificio haga deslizar el pequeño obturador situado en la esquina izquierda del disco:

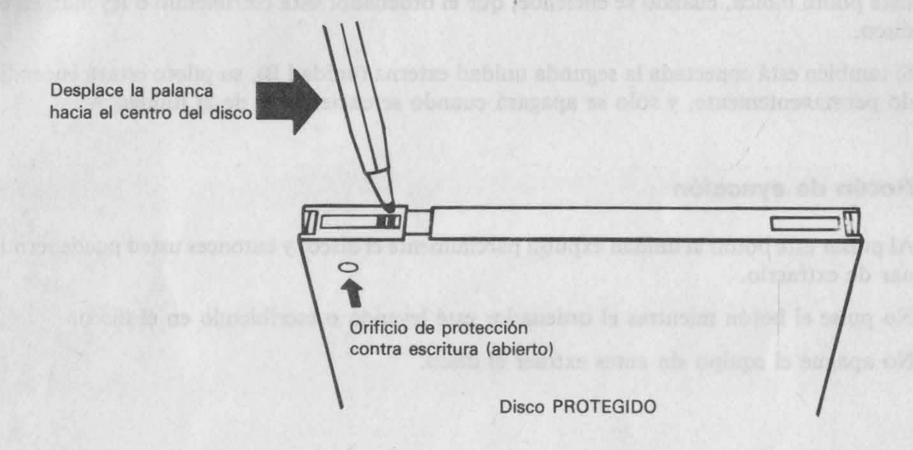
Orificio de protección contra escritura (abierto)
Obturador abierto
Protección activada
PROTEGIDO
Obturador cerrado
Protección desactivada
NO PROTEGIDO

Para cerrar el orificio haga deslizar el obturador en sentido contrario.

En los discos de otros fabricantes el mecanismo consiste en una pequeña palanca de plástico situada en una ranura que tienen en la esquina izquierda:



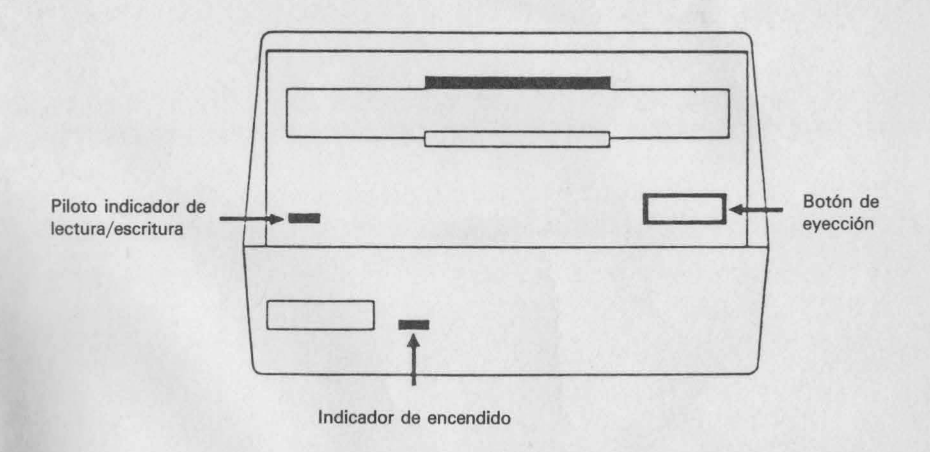
Para abrir el orificio en los discos de este tipo, desplace la palanca hacia el centro del disco, ayudándose con la punta de un bolígrafo u objeto similar:



Observe que, cualquiera que sea el mecanismo de obturación, el efecto es siempre el mismo: el disco queda protegido cuando el orificio está abierto.

Después de introducido el disco

En la cara frontal de la unidad de disco hay un piloto rojo y un botón:



Piloto indicador de lectura/escritura

Este piloto indica, cuando se enciende, que el ordenador está escribiendo o leyendo en el disco.

Si también está conectada la segunda unidad externa (unidad B), su piloto estará encendido permanentemente, y sólo se apagará cuando se encienda el de la unidad A.

Botón de eyección

Al pulsar este botón la unidad expulsa parcialmente el disco, y entonces usted puede terminar de extraerlo.

No pulse el botón mientras el ordenador esté leyendo o escribiendo en el disco.

No apague el equipo sin antes extraer el disco.

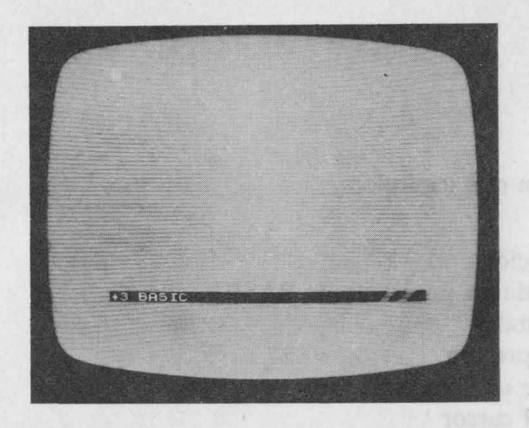
Introducción a +3 BASIC

Temas tratados en este capítulo:

- El editor
- El menú de edición
- Renumeración de un programa de BASIC
- Cambio de pantalla
- Listado por impresora
- Introducción de un programa
- Movimiento del cursor
- Ejecución de un programa
- Órdenes e instrucciones
- Grabación de programas en cinta
- Verificación de programas de cinta
- Carga de programas desde la cinta
- Formateado de un disco
- Grabación de programas en disco
- Nombres de fichero
- Catálogo del disco
- Carga de programas desde el disco
- Mensajes de error

El **+2A** dispone de un avanzado *editor* que podemos utilizar para crear, modificar y ejecutar programas de BASIC. Para activar el editor, seleccione la opción **+3 BASIC** en el menú de presentación utilizando las teclas del cursor y la tecla [INTRO]. (Si no sabe cómo seleccionar una opción del menú, consulte el Capítulo 2.)

En la pantalla se debería ver lo siguiente:



En relación con esta pantalla hay que observar tres cosas:

Primera, que en el extremo superior izquierdo hay un pequeño rectángulo parpadeante, cuyo color alterna entre el blanco y el azul: es el *cursor*. Si pulsa algunas letras en el teclado, éstas aparecerán en la pantalla en la posición del cursor.

Segunda, que en la parte inferior de la pantalla hay una barra negra. Recibe el nombre de *barra de información* porque indica qué parte del software incorporado al **+2A** está siendo utilizada. En este momento dicha barra indica '+3 BASIC', ya que ése es el nombre del editor. (Si ha leído la 'Introducción' de este manual, recordará que dijimos que el **+2A** está provisto de la misma versión de BASIC que el **+3**.)

El último punto a tener en cuenta por el momento es la pantalla pequeña, que se encuentra situada entre la barra de información y el extremo inferior de la pantalla y que generalmente está en blanco. Esta pantalla pequeña sólo tiene espacio para dos líneas de texto y suele ser utilizada por el **+2A** cuando éste detecta un error y necesita escribir un informe para comunicarlo. No obstante, también tiene otras aplicaciones, que describiremos más adelante.

Ahora pulse la tecla [EDIT]. Observará que ocurren dos cosas: se borra el cursor y aparece un nuevo menú, llamado *menú de edición*:



Las opciones del menú de edición se seleccionan de la misma manera que las del menú de presentación (utilizando las teclas del cursor y la tecla [INTRO]).

Veamos esas opciones una por una.

+3 BASIC Esta opción cancela el menú de edición y restituye el cursor. Aunque a primera vista no parezca muy útil, nos permite volver al programa actual, sin alterarlo, si pulsamos [EDIT] accidentalmente.

Renumerar Los programas de BASIC utilizan números de línea para determinar el orden en que las instrucciones han de ser ejecutadas. Estos números (que pueden ser cualquier entero desde el 1 al 9999) debe usted introducirlos al principio de cada línea de programa que escriba. Al seleccionar la opción **Renumerar,** BASIC renumera las líneas de forma que la primera sea la 10 y los números sucesivos vayan aumentando de 10 en 10. Las referencias a números de línea que estén incluidas en las instrucciones del programa (por ejemplo, tras **GO TO, GO SUB, LINE, RESTORE, RUN** y **LIST**) resultan correctamente renumeradas.

Si por alguna razón no es posible renumerar, quizá porque no hay ningún programa en la memoria del ordenador, o porque la operación generaría números de línea mayores que 9999, el **+2A** emite un pitido de tono grave y el menú desaparece.

En la Sección 33 del Capítulo 8 explicaremos una forma de renumerar los programas eligiendo tanto el número de la primera línea como el salto entre líneas.

Pantalla Esta opción lleva el cursor a la zona más pequeña (inferior) de la pantalla y permite que las líneas de BASIC sean introducidas y editadas allí. Esto será útil sobre todo cuando estemos trabajando con gráficos (véase el Capítulo 8, Sección 17), ya que lo que hagamos en la pantalla inferior no interferirá con la pantalla superior. Para volver a esta última (lo cual se puede hacer en cualquier momento durante la edición), seleccione de nuevo la opción **Pantalla** en el menú de edición.

Imprimir Si hay una impresora conectada, esta opción imprimirá un listado del programa actual. Cuando concluya este listado, el menú desaparecerá y volverá el cursor. Si por alguna razón el ordenador no puede imprimir (por ejemplo, porque la impresora se encuentra fuera de línea o está desconectada), pulsando dos veces la tecla [BREAK] se vuelve al editor.

Salida Esta opción conduce de nuevo al menú de presentación (el **+2A** retiene en la memoria el programa con el que se estuviera trabajando). Para volver al programa se selecciona la opción **+3 BASIC** en el menú de presentación.

Si selecciona la opción **48 BASIC** en el menú de presentación (o si apaga o reinicializa con el botón **REINIC** el ordenador), perderá cualquier programa que se encuentre en la memoria. (Sin embargo, sí puede utilizar la opción **Calculadora** del menú de presentación sin perder el programa actual.)

Reinicialice el ordenador y seleccione la opción **+3 BASIC**. Después escriba la siguiente línea. Según los vaya escribiendo, los caracteres aparecerán en la pantalla (un *carácter* es una letra, un número, un espacio, etc.). El signo igual (=) se obtiene pulsando la tecla [L] en combinación con [SIMB].

Ahora escriba esta línea:

10 for f=1 to 100 step 10

y a continuación pulse [INTRO]. Suponiendo que lo haya escrito todo correctamente, el **+2A** habrá reescrito la línea con las palabras **FOR** y **STEP** en letras mayúsculas:

10 FOR f=1 TO 100 STEP 10

El **+2A** habrá emitido también un pitido breve y llevado el cursor al principio de la siguiente línea.

Si la línea permanece en minúsculas y oye usted un pitido de tono grave, esto indica que ha escrito algo equivocado. Observe también que el color del cursor cambia a rojo cuando se detecta un error, y que el ordenador no acepta la línea mientras usted no la corrija. Utilice las teclas del cursor para llevar el cursor hasta el lugar de la línea que hay que corregir y escriba los caracteres que quiera insertar, o utilice la tecla [BORRAR] para suprimir los caracteres sobrantes. Cuando haya terminado de corregir la línea, pulse [INTRO].

Ahora escriba la siguiente línea:

20 plot 0,0:draw f,175:plot 255,0:draw −f,175 (pulse [INTRO])

[El signo de dos puntos (:) se obtiene pulsando la tecla de la [Z] en combinación con [SIMB]; el signo menos (−), con [SIMB] y [J].]

En este momento la pantalla debe estar mostrando lo siguiente:

```
10 FOR f=1 TO 100 STEP 10
20 PLOT 0,0: DRAW f,175: PLOT
  255,0: DRAW −f,175
```

No se preocupe por el hecho de que al llegar al borde derecho de la pantalla el texto «invada» la siguiente línea; el ordenador se encarga de pasar de una línea a otra cuando es necesario y de alinear el texto de forma que pueda ser leído más fácilmente. A diferencia de lo que ocurre con la máquina de escribir, en el ordenador no hay que hacer nada especial cuando se llega al final de una línea de la pantalla, ya que el **+2A** sabe cuándo ocurre esto y lleva el cursor al principio de una nueva línea.

La última línea de este programa es:

30 next f (pulse [INTRO])

Los números que hemos puesto al principio de cada línea son los *números de línea* y sirven para identificarlas. La línea que acabamos de introducir es la 30; el cursor debería encontrarse ahora debajo de ella y a la izquierda. Como ejercicio, vamos a editar la línea 10 para poner **255** en lugar de **100**. Pulse cuatro veces la tecla de 'cursor arriba', [↑]. El cursor sube a la línea 10. Ahora pulse [→] para poner el cursor inmediatamente a la derecha del **100**. Pulse tres veces [BORRAR] y verá cómo desaparece el número. Finalmente, escriba **255** y pulse [INTRO]. La línea 10 del programa ha quedado de esta forma:

```
10 FOR f=1 TO 255 STEP 10
```

Además, el ordenador ha abierto una línea en blanco para que podamos escribir en ella. Escriba:

run (pulse [INTRO])

Esté atento a lo que ocurra. Para empezar, la barra de información y las líneas del programa desaparecen de la pantalla, ya que el editor de +3 BASIC se prepara para ceder el control al programa que acabamos de escribir. A continuación el programa se pone en marcha, dibuja un atractivo diseño y termina con el informe:

0 OK, 30:1

No se preocupe por el significado de este informe.

Pulse [INTRO]. La pantalla se borrará y reaparecerá la barra de información, así como el listado del programa. Esto dura aproximadamente un segundo, durante el cual el **+2A** deja desatendido el teclado, así que no se moleste en escribir nada mientras todo esto ocurre.

Ya hemos realizado la mayor parte de las operaciones necesarias para programar y utilizar un ordenador. En primer lugar, le hemos dado al **+2A** una lista de instrucciones. Las *instrucciones* le dicen al ordenador qué tiene que hacer y cómo debe hacerlo (por ejemplo, **30 NEXT f**). Por otra parte, las instrucciones van precedidas de un número de línea y, cuando usted las escribe, el ordenador las almacena, en lugar de obedecerlas inmediatamente. Finalmente, le dimos al **+2A** la orden **run** para ejecutar el programa que tenía almacenado en la memoria.

Las *órdenes* son similares a las instrucciones, pero no tienen número de línea y el **+2A** las obedece inmediatamente, tan pronto como pulsamos [INTRO]. En general, cualquier instrucción puede ser utilizada como una orden, y viceversa; depende de las circunstancias. Toda orden o instrucción debe contener al menos una palabra clave. Las *palabras clave* constituyen el vocabulario del ordenador, y muchas de ellas necesitan *parámetros*. Por ejemplo, en la orden **DRAW 40,200**, **DRAW** es la palabra clave, mientras que **40** y **200** son los parámetros (que le dicen al ordenador dónde, exactamente, debe realizar el dibujo). Todo lo que el ordenador haga en BASIC se atendrá a estas reglas.

Ahora pulse [EDIT] y seleccione la opción **Pantalla**. El editor desplaza el listado del programa a la zona inferior de la pantalla y se deshace de la barra de información. Sólo es visible la línea 10, ya que el resto del programa está escondido fuera de esta zona de la pantalla (compruébelo subiendo y bajando el cursor).

Pulse [INTRO] y escriba:

run (pulse [INTRO])

El programa es ejecutado exactamente igual que antes. Pero esta vez, si pulsa [INTRO] al concluir la ejecución, la pantalla no se borrará y usted podrá subir y bajar el listado del programa (utilizando las teclas [↑] y [↓]) sin alterar la pantalla superior. Quizás piense que si pulsa ahora [EDIT] para obtener el menú de edición se estropeará la imagen dibujada. Pues no; el **+2A** recuerda lo que hay tras el menú de edición y lo recupera cuando oculta el menú.

Para comprobar que el editor realmente funciona en la parte inferior de la pantalla, pulse [INTRO] y cambie la línea 10 por:

10 FOR f=1 TO 255 STEP 7

Para ello debe llevar el cursor hasta el final de la línea 10 (justamente a la derecha de **STEP 10**), pulsar dos veces [BORRAR], escribir **7** y pulsar [INTRO].

Ahora escriba:

go to 10 (pulse [INTRO])

Las palabras clave **go to** ordenan al **+2A** que no borre la pantalla antes de iniciar la ejecución del programa. Esta versión modificada realiza un dibujo ligeramente distinto, superpuesto al antiguo. Si lo desea, puede continuar editando el programa para añadir cuantos diseños quiera.

Advertencia. Cuando utilice la pantalla inferior, no intente editar instrucciones que tengan una longitud mayor que dos líneas de pantalla, ya que si el editor encuentra una instrucción que tiene su principio o su fin fuera de la pantalla, puede confundirse. (Esto mismo es válido para la pantalla superior, aunque, desde luego, en ese caso no es probable que tal limitación cause problemas, pues la pantalla es mucho más amplia.)

Cuando esté escribiendo, quizá note que las teclas numéricas, al utilizarlas conjuntamente con [MAYUSCULAS], hacen cosas extrañas: [MAYUSCULAS] con [5], [6], [7] y [8] mueve el cursor; [MAYUSCULAS] con [1] invoca el menú de edición; [MAYUSCULAS] con [0] borra un carácter; [MAYUSCULAS] con [2] es equivalente a [BLOQ MAYS]; [MAYUSCULAS] con [9] selecciona el modo gráfico. Todas estas funciones están disponibles utilizando las teclas dedicadas a tal fin en el **+2A**, y no hay ninguna razón por la que sea preferible emplear estas alternativas.

Grabación y carga de programas

En la sección anterior hemos aprendido a introducir un programa en la memoria del ordenador a través del teclado. Este trabajo mecanográfico es inevitable cuando se acaba de diseñar el programa, pero ¿qué ocurre si apagamos el ordenador y queremos usar el mismo programa el día siguiente? ¿Tendremos que volver a escribirlo? Evidentemente, no; el magnetófono incorporado en el **+2A** nos permite grabar el programa (o sea, copiarlo desde la memoria hacia la cinta) para después cargarlo (es decir, copiarlo desde la cinta hacia la memoria). Esto significa que podemos escribir en el teclado un programa, grabarlo en la cinta y después apagar tranquilamente el **+2A**, pues sabemos que en cualquier momento podremos volver a encender el ordenador y cargar el programa desde la cinta hacia la memoria.

Así pues, la última parte de este capítulo está dedicada a estas dos importantes operaciones (**grabación** y **carga** de programas). (Si ha conectado una unidad de disquete externa, pase a la sección titulada 'Operaciones sencillas con los discos', donde explicaremos la grabación y carga de programas de disco.).

Para practicar, vamos a grabar el programa:

```
10 FOR f=1 TO 100 STEP 10
20 PLOT 0,0: DRAW f,175: PLOT
   255,0: DRAW -f,175
```

que habrá quedado en la memoria desde el ejercicio anterior. (Para comprobarlo pulse INTRO, escriba

```
list
```

y vuelva a pulsar INTRO. Si el programa ya no está en la memoria del ordenador, encienda el **+2A**, seleccione la opción + **BASIC** y vuelva a escribir el programa.)

Aunque para grabar programas vale cualquier cinta de cassette, son preferibles las de bajo nivel de ruido.

Grabación de programas en cinta

Cuando tenemos conectada la unidad de disquete externa, el **+2A** da por supuesto que el medio de almacenamiento que queremos usar es el disco. En tal caso, para grabar los programas en cinta debemos dar previamente la orden:

```
save "t:"
```

(Esta orden no es necesaria si no está conectada la unidad de disquete.)

Los programas que grabemos en la cinta deben estar identificados con un *nombre de fichero*. Para el programa que vamos a grabar en este ejemplo es adecuado el nombre 'mi dibujo'; así pues, escriba:

```
save "mi dibujo"
```

y pulse INTRO.

(Los nombres de fichero de cinta admiten hasta 10 caracteres; se puede usar caracteres cualesquiera, e incluso espacios.)

En respuesta a esta orden, el **+2A** emite el mensaje:

```
PREPARE LA CINTA Y PULSE INTRO
```

Primero llevaremos a cabo una grabación simulada para que usted pueda ver lo que ocurrirá cuando más tarde grabemos el programa realmente. Esta vez, pues, **no** pulse los botones de grabación y marcha del magnetófono, sino sólo una tecla del ordenador (por ejemplo, INTRO), y observe el borde de la pantalla. Verá las siguientes pautas de rayas de colores:

- Cinco segundos de rayas de color rojo y cyan desplazándose lentamente hacia arriba, seguidas por una cortísima irrupción de rayas azules y amarillas.
- Una pausa corta.
- Dos segundos de rayas rojas y cyan de nuevo, seguidas por una corta irrupción de rayas azules y amarillas.

Al tiempo que aparecen las rayas en la pantalla, también se puede oír el «sonido» de los datos a través del altavoz del televisor.

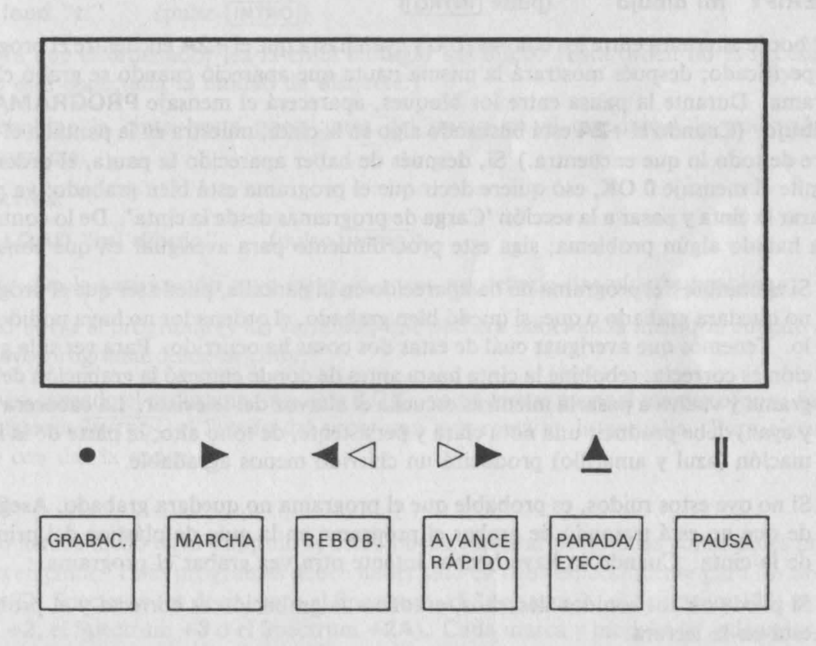
Siga probando la orden **SAVE** anterior (sin poner en marcha el magnetófono) hasta que pueda reconocer esas pautas. Lo que está sucediendo en realidad es que la información está siendo grabada en dos *bloques*, y ambos tienen una «cabecera» (que corresponde a las rayas rojas y cyan) a la que sigue la información propiamente dicha (que corresponde a las rayas azules y amarillas). El primero es un bloque preliminar que contiene el nombre y alguna otra información sobre el programa; el segundo es el programa con sus variables. La pausa entre ellos es sólo un hueco, sin ningún otro significado.

Ahora grabemos realmente el programa en la cinta:

1. Haga avanzar o retroceder la cinta hasta una zona que esté libre o en la que usted haya decidido grabar.
2. Escriba:

   SAVE "mi dibujo"

3. Obedezca el mensaje **'PREPARE LA CINTA Y PULSE INTRO'**.



4. Observe que la pantalla se comporta como vimos antes. Cuando el **+2A** haya terminado de grabar (mensaje **0 OK**), pulse el botón de parada del magnetófono.

Verificación de programas de cinta

Después de grabar un programa, ya se puede apagar tranquilamente el ordenador, o reinicializarlo, o bien comenzar un programa nuevo, sabiendo que podremos volver a cargar el programa siempre que haga falta. No obstante, antes de borrarlo de la memoria del ordenador, debemos comprobar que el proceso de grabación ha funcionado correctamente. Podemos comparar la señal que ha quedado grabada en la cinta con el programa que todavía está en la memoria usando la orden **VERIFY**:

1. Si tiene conectada una unidad de disquete externa y quiere verificar un programa grabado en cinta, dé la orden:

   load "t:" (pulse [INTRO])

   (Esta orden no es necesaria si no está conectada la unidad de disquete.)

2. Rebobine la cinta hasta poco antes del punto en el que inició la grabación del programa.

3. Escriba:

   VERIFY "mi dibujo" (pulse [INTRO])

   El borde alternará entre los colores rojo y cyan hasta que el **+2A** encuentre el programa especificado; después mostrará la misma pauta que apareció cuando se grabó el programa. Durante la pausa entre los bloques, aparecerá el mensaje **PROGRAMA: mi dibujo**. (Cuando el **+2A** está buscando algo en la cinta, muestra en la pantalla el nombre de todo lo que encuentra.) Si, después de haber aparecido la pauta, el ordenador emite el mensaje **0 OK**, eso quiere decir que el programa está bien grabado; ya puede parar la cinta y pasar a la sección 'Carga de programas desde la cinta'. De lo contrario, ha habido algún problema; siga este procedimiento para averiguar en qué consiste:

   - Si el nombre del programa no ha aparecido en la pantalla, puede ser que el programa no quedara grabado o que, si quedó bien grabado, el ordenador no haya podido leerlo. Tenemos que averiguar cuál de estas dos cosas ha ocurrido. Para ver si la grabación es correcta, rebobine la cinta hasta antes de donde empezó la grabación del programa y vuelva a pasarla mientras escucha el altavoz del televisor. La cabecera (rojo y cyan) debe producir una nota clara y persistente, de tono alto; la parte de la información (azul y amarillo) producirá un chirrido menos agradable.

   - Si no oye estos ruidos, es probable que el programa no quedara grabado. Asegúrese de que no está tratando de grabar el programa en la guía de plástico del principio de la cinta. Cuando lo haya hecho, intente otra vez grabar el programa.

   - Si puede oír los sonidos descritos, entonces la grabación es correcta y el problema está en la lectura.

   - Quizá haya escrito mal el nombre del programa al grabarlo; en tal caso, cuando el **+2A** encuentra el programa, muestra en la pantalla el nombre incorrecto. También

puede haber escrito mal el nombre tras la orden **VERIFY**, en cuyo caso el ordenador ignorará el programa que está bien grabado y continuará buscando el nombre erróneo, provocando destellos de rojo y cyan según avanza.

- Si realmente hay un error en la cinta, el **+2A** mostrará el mensaje **R ERROR CARGANDO CINTA**, lo cual significa en este caso que ha fracasado la operación de verificación. Un pequeño defecto de la cinta misma (que podría ser casi inaudible en una grabación musical) puede causar estragos en un programa de ordenador. Trate de grabar el programa de nuevo, tal vez en una parte diferente de la cinta.

Carga de programas desde la cinta

Si ha grabado el programa y lo ha verificado con éxito, reinicialice el ordenador y cargue el programa por el siguiente procedimiento:

1. Pulse el botón **REINIC** y seleccione la opción **Cargador** en el menú de presentación.
2. Si tiene conectada una unidad de disquete externa, dé la orden:

   load "t:" (pulse INTRO)

   para que el ordenador lea la cinta en lugar del disco. (Esta orden no es necesaria si no está conectada la unidad de disquete.)
3. Rebobine la cinta hasta poco antes del punto en el que inició la grabación del programa.
4. Escriba:

   LOAD "mi dibujo" (pulse INTRO)

Puesto que la verificación tuvo éxito, la carga no debería dar ningún problema.

LOAD borra el programa (y las variables) que pudiera haber en la memoria cuando carga el nuevo programa desde la cinta.

Una vez cargado el programa (mensaje **0 OK**), ya podemos parar el magnetófono. Si ahora pulsamos INTRO, el listado del programa aparecerá en la pantalla. Para ejecutarlo basta con dar la orden:

run

Como hemos dicho en el Capítulo 4, usted puede comprar programas comerciales pregrabados en cinta. Tales programas deben haber sido escritos especialmente para los ordenadores ZX Spectrum (es decir, para el Spectrum, el Spectrum +, el Spectrum 128, el Spectrum **+2**, el Spectrum **+3** o el Spectrum **+2A**). Cada marca y modelo de ordenador tiene una forma diferente de almacenar programas, así que con este ordenador no es posible leer las cintas grabadas por o para otros.

Si en la misma cara de la cinta hay varios programas, cada uno tendrá un nombre. En la orden **LOAD** se puede especificar qué programa se desea cargar: por ejemplo, si el que queremos cargar se llama 'aeropuerto', podemos escribir:

LOAD "aeropuerto"

La orden **LOAD ""** significa 'carga el primer programa que encuentres en la cinta'. Esto puede ser útil si usted no recuerda el nombre con el que grabó el programa.

La opción **Cargador** del menú de presentación produce el mismo efecto que **LOAD ""**, y es mucho más fácil y rápida de usar (basta con encender el ordenador y pulsar [INTRO]). (Si tiene conectada la unidad de disquete, asegúrese de que está vacía.)

No se puede usar la opción **Cargador** para cargar programas escritos para el Spectrum 48.

Operaciones sencillas con los discos

Si ha conectado una unidad de disquete externa, debe leer el resto de este capítulo para aprender a grabar y cargar programas en disco. Si no, puede pasar al siguiente capítulo.

Para seguir los ejemplos de los siguientes apartados necesitará un disco vacío. **No utilice un disco que contenga programas valiosos que desee conservar.**

Discos y cintas

Aunque usted ya tenga experiencia en la grabación y carga de programas en cinta, hay dos cuestiones importantes que debe recordar al manejar discos:

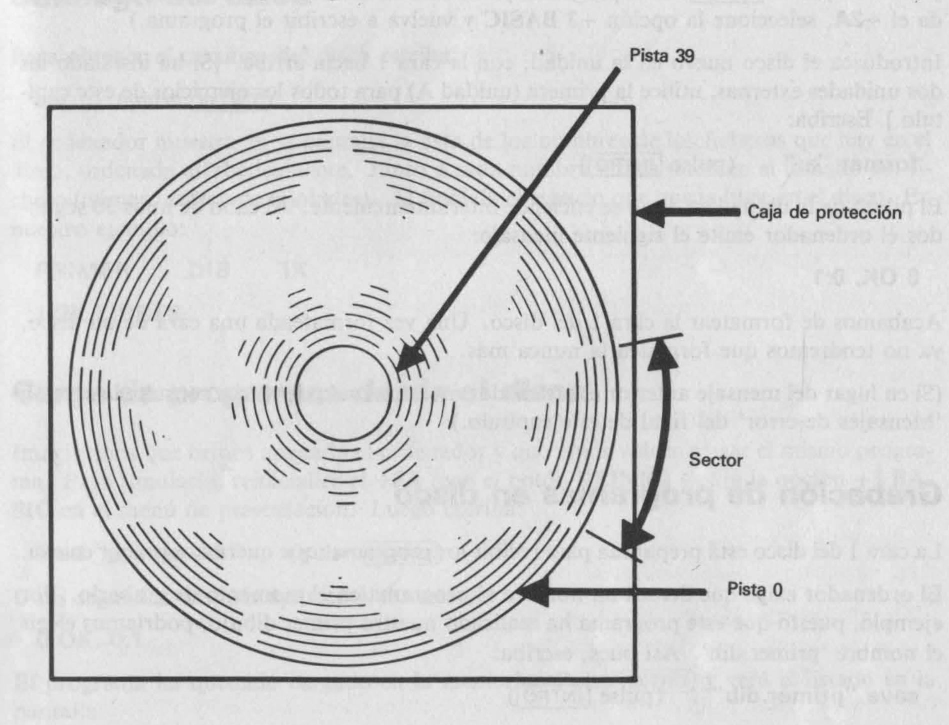
En primer lugar, a diferencia de lo que ocurre con las cintas, para grabar en un disco no basta con sacarlo de la caja e introducirlo en la unidad. Los discos nuevos tienen que ser 'formateados'. **Al formatear un disco usado se borra de forma irrecuperable su contenido.**

En segundo lugar, los nombres que podemos dar a los ficheros de disco tienen que cumplir ciertas reglas. Los nombres de los ficheros de cinta pueden ser bastante largos, e incluso en ocasiones podemos omitirlos. No ocurre así con el sistema de grabación en disco, en el que los nombres de los ficheros tienen que atenerse a normas estrictas (como veremos más adelante, en la sección 'Nombres de fichero').

Formateado de un disco

La operación de formatear un disco consiste en marcar en el disco una especie de estantería, cuyos estantes son los lugares en los que más tarde almacenaremos la información.

Al formatear un disco, éste queda dividido en 360 zonas:



Radialmente el disco se divide en 40 pistas concéntricas, desde la número 0, que es la más externa, hasta la número 39. Cada pista se divide en 9 sectores.

Cada sector puede almacenar 512 bytes; por lo tanto, la capacidad total de cada disco es $40\times9\times512=184320$ bytes (180K). No obstante, 7 de estos 180K están reservados para uso propio del ordenador, de modo que la capacidad disponible para grabar programas es de unos 173K.

Para practicar, vamos a formatear un disco y a grabar en él el programa:

```
10 FOR f=1 TO 100 STEP 10
20 PLOT 0,0: DRAW f,175: PLOT
   255,0: DRAW -f,175
```

que habrá quedado en la memoria desde el ejercicio anterior. (Para comprobarlo pulse INTRO, escriba

```
list
```

y vuelva a pulsar [INTRO]. Si el programa ya no está en la memoria del ordenador, encienda el **+2A**, seleccione la opción **+3 BASIC** y vuelva a escribir el programa.)

Introduzca el disco nuevo en la unidad, con la cara 1 hacia arriba. [Si ha instalado las dos unidades externas, utilice la primera (unidad A) para todos los ejercicios de este capítulo.] Escriba:

format "a:" (pulse [INTRO])

El piloto de la unidad de disco se enciende intermitentemente. Al cabo de unos 30 segundos el ordenador emite el siguiente mensaje:

0 OK, 0:1

Acabamos de formatear la cara 1 del disco. Una vez formateada una cara de un disco, ya no tendremos que formatearla nunca más.

(Si en lugar del mensaje anterior el ordenador muestra cualquier otro, consulte la sección 'Mensajes de error' del final de este capítulo.)

Grabación de programas en disco

La cara 1 del disco está preparada para recibir los programas que queramos grabar en ella.

El ordenador exige que demos un nombre al programa en el momento de grabarlo. Por ejemplo, puesto que este programa ha realizado nuestro primer dibujo, podríamos elegir el nombre 'primer.dib'. Así pues, escriba:

save "primer.dib" (pulse [INTRO])

Al cabo de unos segundos aparecerá el mensaje:

0 OK, 0:1

El programa ha quedado grabado en el disco.

(Si en lugar de este mensaje el ordenador muestra cualquier otro, consulte la sección 'Mensajes de error' del final de este capítulo.)

Nombres de fichero

El nombre del fichero consta de dos partes (*campos*). El primer campo es obligatorio y puede contener hasta 8 caracteres (letras o números, pero no signos de puntuación ni espacios). En nuestro ejemplo, el primer campo era 'primer'.

El segundo campo es opcional. Puede contener hasta 3 caracteres (no espacios ni signos de puntuación). En el ejemplo, el segundo campo era 'dib'.

Si incluimos los dos campos, tenemos que separarlos con un punto.

Catálogo del disco

Para obtener el catálogo del disco escriba:

cat (pulse [INTRO])

El ordenador muestra en la pantalla la lista de los nombres de los ficheros que hay en el disco, ordenada alfabéticamente. Junto a cada nombre indica también el tamaño del fichero (número entero de kilobytes). Al final da el espacio que queda libre en el disco. En nuestro ejemplo:

PRIMER .DIB 1K

172K LIBRES

Carga de programas desde el disco

Imaginemos que hemos apagado el ordenador y queremos volver a usar el mismo programa. Para simularlo, reinicialice el **+2A** (con el botón **REINIC**) y elija la opción **+3 BASIC** en el menú de presentación. Luego escriba:

load "primer.dib" (pulse [INTRO])

Unos segundos después aparecerá el mensaje:

0 OK, 0:1

El programa ha quedado cargado en la memoria. Pulse [INTRO] y verá el listado en la pantalla.

(Si en lugar de este mensaje el ordenador muestra cualquier otro, consulte la sección siguiente, 'Mensajes de error'.)

Una vez cargado el programa, para iniciar su ejecución basta con escribir:

run

y pulsar [INTRO], igual que hacíamos antes.

Mensajes de error

Cualquier error que haya cometido al escribir las órdenes de las secciones anteriores habrá dado como resultado uno de los siguientes *mensajes de error*. De ser así, identifique el mensaje, lea la explicación y actúe según lo recomendado.

UNIDAD A: NO PREPARADA

Lo más probable es que se haya olvidado de introducir el disco en la unidad. De lo contrario, extraiga el disco, vuelva a introducirlo y pulse [R] para reintentar la operación.

DISCO PROTEGIDO

Este mensaje indica que el disco que estamos intentando formatear o grabar tiene abierto el orificio de protección contra escritura. Extraiga el disco, cierre el orificio de protección y reintente la operación.

FICHERO NO ENCONTRADO

Este mensaje indica que hemos tratado de cargar un fichero que no está en esa cara del disco. Extraiga el disco, introdúzcalo con la cara correcta hacia arriba y reintente. Asegúrese de que escribe correctamente el nombre del fichero.

NOMBRE INCORRECTO

Este mensaje indica que estamos intentando cargar o grabar un programa usando un nombre no válido (o sin nombre). Lea la sección 'Nombres de fichero' y reintente.

Ya formateado. Tecla A para abandonar/otra para continuar

El ordenador ha observado que el disco ya está formateado y nos avisa en prevención de que vayamos a reformatearlo por error. En general, cada disco sólo debe ser formateado una vez. Formatearlo por segunda vez sólo tiene interés si se estropea y queremos reciclarlo. Pulse [A] para no formatear de nuevo el disco, o cualquier otra tecla para formatearlo.

Nota. Si no pulsa [A], el proceso continuará y borrará completamente el disco.

Si observa que un disco concreto (o una de sus caras) vuelve a estropearse después de reformatearlo, es muy probable que tenga un defecto físico; lo aconsejable entonces es no volver a utilizar ese disco (o al menos esa cara).

Algunos de estos mensajes van seguidos de una línea que ofrece tres opciones:

¿REINTENTAR, IGNORAR O CANCELAR?

- Si se pulsa [R] (tras tomar las medidas correctoras oportunas) el ordenador vuelve a intentar la operación.
- Si se pulsa [I], el ordenador ignora la causa del error y continúa; no se debe elegir esta opción si no se sabe qué consecuencias va a tener.
- Si se pulsa [C], el ordenador abandona la operación (y entonces puede aparecer un nuevo mensaje).

Más información sobre el sistema de disco

En la Sección 20 del Capítulo 8 puede encontrar información más detallada sobre el manejo de ficheros de disco y de cinta y sobre el disco de RAM. El sistema operativo de disco del **+2A** (+3DOS) está descrito en la Sección 27 del Capítulo 8.

UNIDAD A: NO PREPARADA

Lo más probable es que se haya olvidado de introducir el disco en la unidad. De lo contrario, extraiga el disco, [illegible]

DISCO PROTEGIDO

Este mensaje indica que el disco que estamos intentando formatear o grabar tiene abierto el orificio de protección contra escritura. Extraiga el disco, cierre el orificio de protección y reintente la operación.

FICHERO NO ENCONTRADO

Este mensaje indica que hemos tratado de cargar un fichero que no está en el directorio del disco. Extraiga el disco, introdúzcalo con la cara correcta hacia arriba y reinténtelo. Asegúrese de que escribe correctamente el nombre del fichero.

NOMBRE INCORRECTO

Este mensaje indica que estamos intentando cargar o grabar un programa usando un nombre no válido (o sin nombre). Lea la sección "Nombres de fichero" y reinténtelo.

Ya formateado. Teclee A para abandonar/otra para continuar

El ordenador ha observado que el disco ya está formateado y nos avisa en prevención de que vayamos a reformatearlo por error. En general, cada disco sólo debe ser formateado una vez. Formatearlo por segunda vez sólo tiene interés si se estropea y queremos reciclarlo. Pulse [A] para no formatear de nuevo el disco, o cualquier otra tecla para formatearlo.

Nota: Si no pulsa [A], el proceso continuará y borrará completamente el disco.

Si observa que un disco concreto (o una de sus caras) vuelve a estropearse después de reformatearlo, es muy probable que tenga un defecto físico; lo aconsejable entonces es no volver a utilizar ese disco (o al menos esa cara).

Algunos de estos mensajes van seguidos de una línea que ofrece tres opciones:

¿REINTENTAR, IGNORAR O CANCELAR?

- Si se pulsa [R] (tras tomar las medidas correctoras oportunas) el ordenador vuelve a intentar la operación.
- Si se pulsa [I], el ordenador ignora la causa del error y continúa; no se debe elegir esta opción si no se sabe qué consecuencias va a tener.
- Si se pulsa [C], el ordenador abandona la operación (y entonces puede aparecer un nuevo mensaje).

Utilización de 48 BASIC

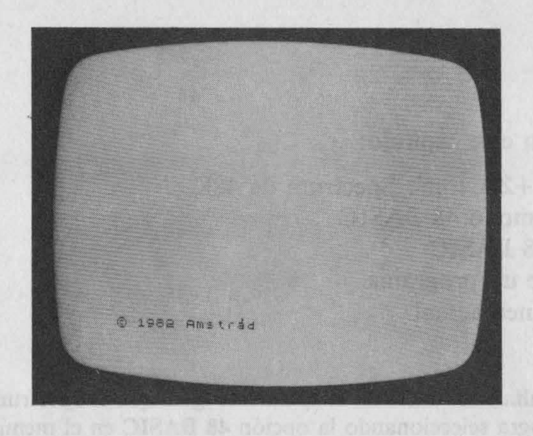
Temas tratados en este capítulo:

- Utilización del **+2A** como Spectrum de 48K
- Activación del modo 48 BASIC
- El teclado en 48 BASIC
- Introducción de un programa
- Edición de la línea actual

El **+2A** tiene la facultad de funcionar exactamente igual que el Spectrum de 48K (o Spectrum +). Esto se logra seleccionando la opción **48 BASIC** en el menú de presentación. En este modo no se puede aprovechar las características más avanzadas del **+2A**, como son las unidades de disco, la memoria adicional, el editor de pantalla completa, el sonido en varios canales, los interfaces **RS232/MIDI/AUX** y el disco de RAM. Sin embargo, sí funcionarán los zócalos **JOYSTICK 1** y **JOYSTICK 2**.

El modo 48 BASIC ha sido incluido sólo por razones de compatibilidad: no hay ninguna ventaja en utilizar este modo (en vez de +3 BASIC) para escribir programas, y **no** lo recomendamos. La siguiente información sólo ha sido incluida a modo de referencia y para quienes estén acostumbrados al Spectrum de 48K y quieran utilizar inmediatamente la máquina sin tener que aprender el manejo del editor de +3 BASIC.

En realidad, hay dos métodos para llevar el **+2A** al modo 48 BASIC. El primero consiste en seleccionar la opción **48 BASIC** en el menú de presentación (si no sabe cómo hacerlo, consulte el capítulo 2). Una vez seleccionada esta opción, verá lo siguiente en la pantalla:



El segundo método permite entrar en el modo 48 BASIC mientras se está editando un programa en +3 BASIC. Para ello, estando en el editor de +3 BASIC, escriba:

spectrum (pulse [INTRO])

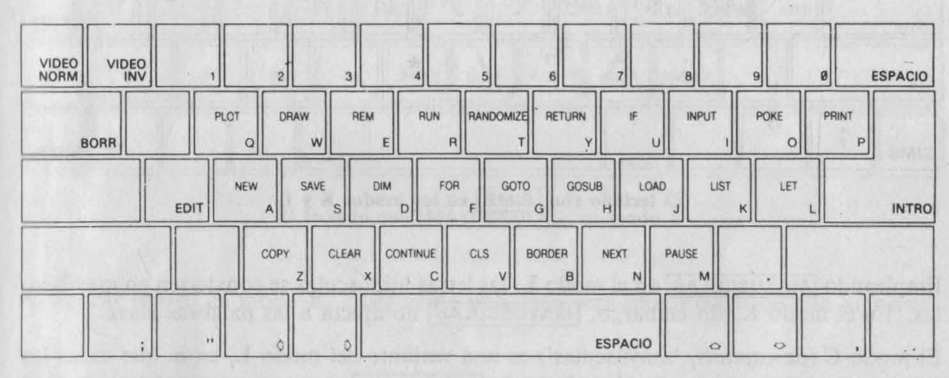
El **+2A** responderá con el mensaje **OK** y habrá cambiado al modo 48 BASIC conservando el programa actualmente almacenado en la memoria. Una vez en 48 BASIC, ya no hay forma de volver a +3 BASIC, salvo reinicializando el **+2A** (o apagándolo y volviendo a encenderlo).

La principal diferencia entre las dos versiones de BASIC se encuentra en la forma de introducir y editar programas. Observe también que en +3 BASIC los códigos **SPECTRUM** y **PLAY** han reemplazado a los caracteres gráficos definibles por el usuario para las teclas **T** y **U** (valores 163 y 164).

En el modo 48 BASIC el teclado funciona de la siguiente manera:

Todos los operadores, órdenes y funciones de BASIC están disponibles *directamente* desde el teclado y no es necesario escribirlos letra a letra. A fin de acomodar todas estas funciones y órdenes, algunas teclas tienen cinco o más significados diferentes, que se obtienen en parte modificando el *estado* de las teclas (es decir, pulsando las teclas en combinación con [MAYUSCULAS] o [SIMB]), y en parte poniendo la máquina en diferentes *modos de teclado*. El cursor parpadeante contiene una letra (**K**, **L**, **C**, **E** o **G**) para indicar qué modo está activado en cada momento.

El modo **K** (de *keywords*, 'palabras clave') reemplaza automáticamente al modo **L** (de 'letras') cuando el ordenador está esperando una orden o una línea de programa (en vez de la introducción de datos). Por la posición que ocupa el cursor en la línea, el **+2A** sabe si debe esperar un número de línea o una palabra clave. El modo **K** se activa al principio de la línea, después del signo de dos puntos (:) (salvo cuando éste forma parte de una cadena literal) y tras la palabra clave **THEN**. Siempre que aparezca el cursor **K**, la siguiente tecla que se pulse será interpretada como palabra clave o como número, según se muestra en la siguiente figura:



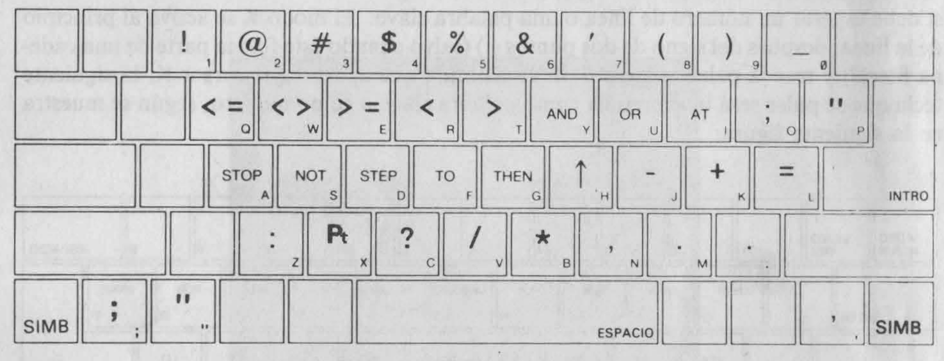
El teclado en modo K

El modo **L** (de 'letras') es el que está activo normalmente (salvo cuando lo está el modo **K**). Siempre que aparezca el cursor **L**, la siguiente tecla pulsada será interpretada de acuerdo con la leyenda que está grabada en la propia tecla; es decir,



El teclado en modo L

En ambos modos, **K** y **L**, la combinación de las diferentes teclas con SIMB es interpretada de la siguiente manera:

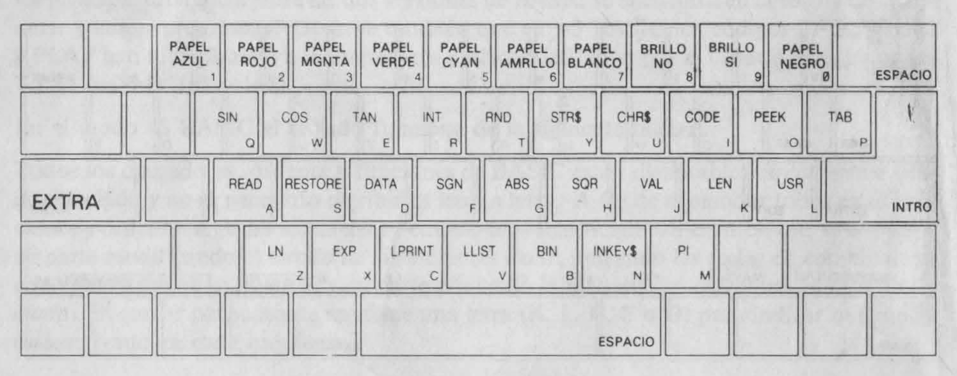


El teclado con SIMB en los modos K y L

Empleando MAYUSCULAS en el modo **L**, las letras minúsculas se convierten en mayúsculas. En el modo **K**, sin embargo, MAYUSCULAS no afecta a las palabras clave.

El modo **C** (de *capitals*, 'mayúsculas') es una variante del modo **L**, en la que todas las letras aparecen como mayúsculas. La tecla BLOQ MAYS sirve para pasar del modo **L** al **C**, y viceversa.

El modo **E** (de 'extra') se utiliza para obtener diversos caracteres, principalmente códigos de palabras clave. Este modo se activa pulsando la tecla EXTRA y sólo afecta al siguiente carácter (o pulsación de una tecla). Siempre que aparezca el cursor **E**, la siguiente pulsación será interpretada de esta manera:



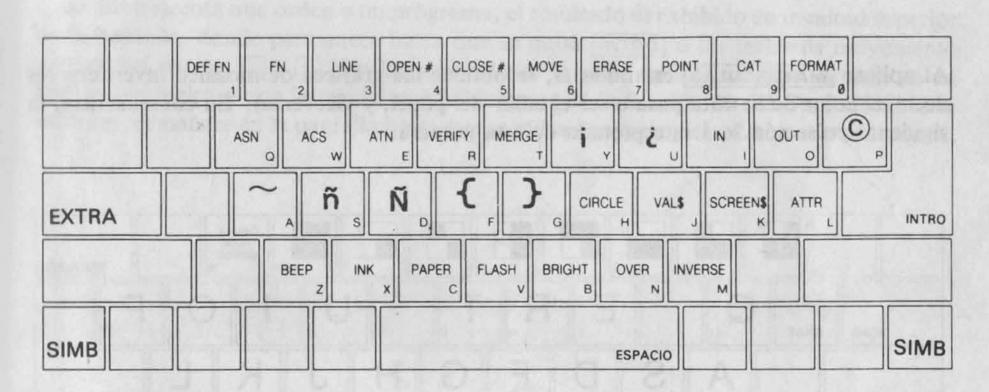
El teclado en modo E

Al pulsar MAYUSCULAS en modo **E**, la siguiente pulsación será interpretada de esta manera:



El teclado con MAYUSCULAS en modo E

Al aplicar SIMB en modo **E**, la siguiente pulsación será interpretada de esta manera:



El teclado con SIMB en modo E

El modo **G** (de 'gráficos') se obtiene pulsando [GRAF] y permanece activado hasta que se vuelve a pulsar la misma tecla (o la [9]). En este modo, cada tecla numérica proporciona un gráfico de mosaico, y cada tecla alfabética (a excepción de [V], [W], [X], [Y] y [Z]) da un gráfico definible por el usuario, el cual, mientras no haya sido definido, tendrá forma idéntica al de la letra mayúscula. Siempre que aparezca el cursor **G**, la siguiente pulsación será interpretada de esta manera:



El teclado en modo G

Al aplicar [MAYUSCULAS] en modo **G**, se obtiene los gráficos de mosaico invertidos (es decir, el color de la tinta pasa a ser el color del papel, y viceversa). En consecuencia, la siguiente pulsación será interpretada de esta manera:



El teclado con [MAYUSCULAS] en modo G

Observaciones generales sobre el teclado

Si se mantiene pulsada una tecla durante más de 2 o 3 segundos, comenzará a repetirse. Todo lo introducido por el teclado aparece en la mitad inferior de la pantalla según se va escribiendo; cada carácter (símbolo sencillo o código de palabra clave) es insertado justamente donde está el cursor. Éste puede ser desplazado a derecha e izquierda mediante las teclas de movimiento horizontal del cursor, [→] y [←], que se encuentran a la izquierda de la barra espaciadora. El carácter que está a la izquierda del cursor se borra con la tecla [BORRAR].

Cuando se pulsa [INTRO], la línea es ejecutada, almacenada como línea de programa o utilizada como entrada de información (para **INPUT**). No obstante, si la línea contiene un *error de sintaxis*, junto al error aparece un signo de interrogación parpadeante.

Según se va introduciendo líneas de programa, en la mitad superior de la pantalla va apareciendo un listado. La última línea introducida se llama *línea actual* y está indicada por el símbolo > a la derecha del número de línea. Cualquier línea del programa puede ser seleccionada como línea actual (con objeto de editarla) utilizando las teclas [↓] y [↑] (que se encuentran a la derecha de la barra espaciadora). Para editar la línea actual así seleccionada se debe pulsar la tecla [EDIT]. (La edición tiene lugar en la parte inferior de la pantalla.)

Cuando se ejecuta una orden o un programa, el resultado es exhibido en la mitad superior de la pantalla, donde permanece hasta que se pulsa [INTRO] o las teclas de movimiento vertical del cursor, [↓] y [↑]. En la parte inferior aparece un informe que consiste en un código (un dígito o una letra) del que hablaremos en la Sección 29 del Capítulo 8. Este informe permanece en la pantalla hasta que se pulsa una tecla y el **+2A** vuelve al modo **K**.

Guía de programación en +3 BASIC

Sección 1
Introducción

Tanto si ya ha leído el Capítulo 6 como si ha venido directamente a éste, debe saber que:

—las *órdenes* son obedecidas inmediatamente;

—las *instrucciones* comienzan con un número de línea y son almacenadas para su uso posterior.

Esta guía de BASIC comienza repitiendo algunas de las cosas tratadas en el Capítulo 6, pero de forma más detallada. Al final de algunas secciones hemos incluido ejercicios: le recomendamos que no los pase por alto, pues muchos de ellos ilustran conceptos que sólo han sido mencionados de pasada en el texto. Écheles un vistazo y trabaje con los que más le interesen, o con los que traten alguna cuestión que el texto no le haya dejado completamente clara.

El teclado



Los caracteres utilizados en el **+2A** no son solamente símbolos simples (letras, dígitos, etc.) sino también códigos de palabras clave, nombres de funciones, etc. Todo debe ser escrito letra por letra, y en la mayor parte de los casos es indiferente hacerlo en mayúsculas o minúsculas. En el teclado hay tres clases de teclas: las de letras y números (llamadas alfanuméricas), las de símbolos (signos de puntuación) y las teclas de control (tales como MAYUSCULAS, BORRAR, etc.).

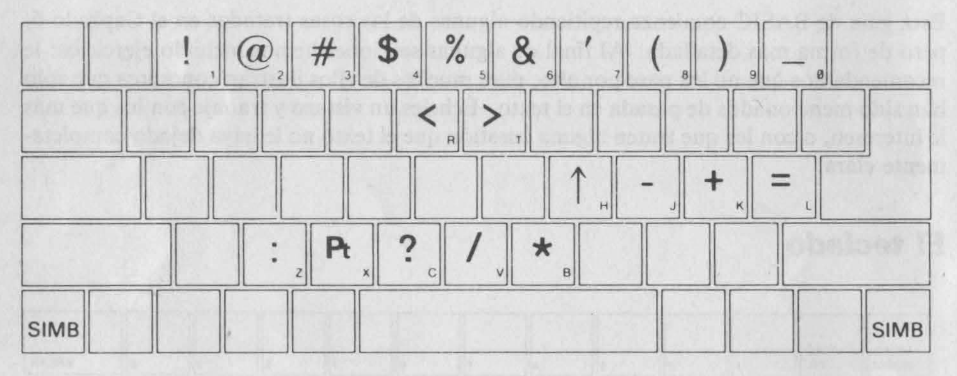
Las teclas alfanuméricas son las más frecuentemente utilizadas en BASIC. Cuando se pulsa una tecla alfabética, en la pantalla aparece una letra minúscula junto con un cuadrado parpadeante (cuyo color alterna entre azul y blanco), llamado *cursor*. Para obtener la mayúscula se debe mantener pulsada la tecla MAYUSCULAS al tiempo que se escribe la letra.

Si desea escribir con mayúsculas continuamente, pulse una vez la tecla BLOQ MAYS: todas las teclas alfabéticas que pulse en lo sucesivo producirán letras mayúsculas. Para volver a las minúsculas, pulse otra vez BLOQ MAYS.

Para escribir los símbolos que aparecen en las teclas alfanuméricas, es decir,

! @ # $ % & ' ( ) _ < > ↑ − + = : Pt ? / *

pulse la tecla correspondiente en combinación con SIMB (véase el diagrama siguiente).



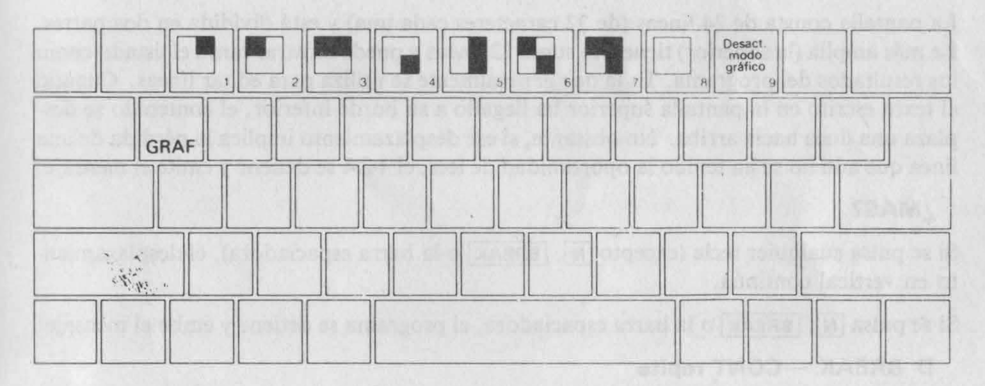
Símbolos disponibles con SIMB

Además se puede obtener los símbolos ¡ ¿ © ˜ ñ Ñ { y } pulsando una vez la tecla EXTRA y pulsando luego una tecla alfabética en combinación con SIMB (véase el diagrama siguiente).



Símbolos disponibles con SIMB en modo EXTRA

Para activar el modo de gráficos, pulse una vez la tecla GRAF. Pulsando las teclas numéricas (excepto 9 y 0) se obtiene los gráficos de mosaico (véase el siguiente diagrama). Pulsando las teclas alfabéticas (excepto T, U, V, W, X, Y y Z) se obtiene los gráficos definidos por el usuario.



Gráficos de mosaico disponibles con GRAF

Combinando MAYUSCULAS con las teclas numéricas citadas se obtiene los mismos gráficos de mosaico, pero con los colores invertidos.

Observaciones generales sobre el teclado

Si se mantiene pulsada una tecla durante más de 2 o 3 segundos, comenzará a repetirse. A medida que se va pulsando teclas, se forma una línea en la pantalla. Dicho sea de paso, por «línea» entendemos una línea de BASIC, la cual puede ocupar varias líneas físicas en la pantalla. La línea puede ser recorrida utilizando las teclas de movimiento del cursor: [←], [→], [↑], [↓]. Si la parte de la línea hacia la que se mueve el cursor se encuentra fuera de la pantalla, entonces el texto se desplazará hacia arriba o hacia abajo para hacerla visible. Cualquier carácter que se escriba será insertado en la posición del cursor. Pulsando [BORRAR] se borra el carácter que está a la izquierda del cursor. En cuanto se pulsa [INTRO], o si se intenta sacar el cursor de la línea, el ordenador comprueba si la línea tiene sentido. Si lo tiene, genera un pitido agudo y obedece la línea inmediatamente, o bien la almacena como parte de un programa. Si la línea contiene un error, el ordenador emite un pitido grave y lleva el cursor a la zona en la que piensa que se encuentra dicho error (además, el color del cursor cambia a rojo). Es imposible salir de una línea que contenga un error: el **+2A** siempre volverá a llevar el cursor a esa línea.

La pantalla

La pantalla consta de 24 líneas (de 32 caracteres cada una) y está dividida en dos partes. La más amplia (la superior) tiene a lo sumo 22 líneas y puede mostrar tanto el listado como los resultados del programa. Es la que generalmente se utiliza para editar líneas. Cuando el texto escrito en la pantalla superior ha llegado a su borde inferior, el contenido se desplaza una línea hacia arriba. No obstante, si ese desplazamiento implica la pérdida de una línea que aún no se ha tenido la oportunidad de leer, el **+2A** se detiene y emite el mensaje:

¿MAS?

Si se pulsa cualquier tecla (excepto [N], [BREAK] o la barra espaciadora), el desplazamiento en vertical continúa.

Si se pulsa [N], [BREAK] o la barra espaciadora, el programa se detiene y emite el mensaje:

D BREAK – CONT repite

La parte más pequeña (inferior) de la pantalla se utiliza para editar programas cortos, introducir datos y órdenes directas (cuando no conviene usar la pantalla superior; por ejemplo, en programas de gráficos) y para exhibir informes.

Introducción de programas

Si el programa que está siendo introducido sobrepasa el tamaño de la pantalla, el **+2A** intentará presentar el área de mayor interés (generalmente, la última línea introducida jun-

to con las cercanas a ella). No obstante, podemos hacer que el ordenador muestre otra área del programa especificándola en la orden:

LIST *xxx*

donde *xxx* es un número de línea. Esta orden hace que el **+2A** exhiba la zona del programa especificada.

Cuando se ejecuta una orden o un programa, el resultado se muestra en la pantalla superior, donde permanece cuando el programa termina (hasta que se pulsa una tecla). Si el programa está siendo editado en la pantalla inferior, los resultados exhibidos en la pantalla superior permanecerán en ella hasta que se escriba sobre ellos, hasta que desaparezcan por efecto del desplazamiento en vertical o hasta que se ejecute una orden **CLS**. La pantalla inferior puede mostrar un informe, precedido de un código (dígito o letra), cuyo significado explicaremos en la Sección 28 de este capítulo. Ese informe permanecerá en la pantalla inferior hasta que se pulse una tecla.

Mientras el **+2A** está ejecutando un programa de BASIC, la tecla [BREAK] es comprobada de vez en cuando; concretamente, al final de cada sentencia, durante la utilización del magnetófono y de la impresora (si se ha conectado estos periféricos) y mientras se interpreta la música. Si el **+2A** detecta que se ha pulsado [BREAK], detiene la ejecución del programa y emite un informe. Entonces se puede editar el programa.

Sección 2
Conceptos sencillos de programación

Temas tratados:

Programas
Números de línea
Edición de programas utilizando las teclas del cursor
RUN, LIST
GO TO, CONTINUE, INPUT, NEW, REM, PRINT
Detención de un programa

Escriba las dos primeras líneas de un programa que, cuando esté completo, calculará y exhibirá en la pantalla la suma de dos números:

```
20 print a
10 let a=10
```

(No olvide pulsar [INTRO] al final de cada línea.) Observe que la pantalla muestra lo siguiente:

```
10 LET a=10
▌
20 PRINT a
```

Como hemos visto antes, puesto que estas líneas comenzaban con números, el **+2A** no las obedeció inmediatamente, sino que las almacenó como líneas de programa. Este ejemplo muestra también que los números de línea controlan el orden en que las líneas del programa van a ser ejecutadas; tal como puede ver en la pantalla, el **+2A** ordena todas las líneas cada vez que introducimos una nueva.

Observe además que, aunque habíamos escrito cada línea con letras minúsculas, el ordenador ha convertido a mayúsculas las palabras clave (**PRINT** y **LET**) en cuanto ha aceptado la línea. De ahora en adelante mostraremos en letras mayúsculas las palabras clave; sin embargo, usted puede continuar escribiéndolas en minúsculas. (Si no recuerda qué son las 'palabras clave', repase el Capítulo 6.)

Hasta ahora sólo hemos introducido un número de los dos que tenemos que sumar, así que escriba:

15 LET b=15 (pulse [INTRO])

Ahora debemos transformar la línea 20 del siguiente modo:

20 PRINT a+b

Podríamos escribir la nueva línea completa, pero es mucho más cómodo situar el cursor justamente detrás de la **a** y luego escribir:

+b (no pulse todavía [INTRO])

La línea debería quedar así:

20 PRINT a+b

Pulse [INTRO]; el cursor se desplaza a la línea de abajo, de forma tal que en la pantalla se ve lo siguiente:

```
10 LET a=10
15 LET b=15
20 PRINT a+b
█
```

Lo que hemos hecho en este programa es *asignar* el valor 10 a la *variable* llamada **a**, y el valor 17 a la variable **b**. Después le hemos pedido al ordenador que escriba (**PRINT**) la suma de los dos valores.

Para ejecutar el programa dé la orden:

RUN (pulse [INTRO])

La suma aparecerá en la pantalla:

25

Ejecute nuevamente el programa y después dé la orden:

PRINT a,b (pulse [INTRO])

El ordenador responde escribiendo:

10 15

Esto demuestra que las variables continúan en la memoria, a pesar de que el programa ya ha terminado.

Errores

Si introducimos una línea por error, v.g.,

12 LET b=8

para borrarla basta con escribir su número y pulsar [INTRO]:

12 (pulse [INTRO])

La línea 12 desaparece del listado y el cursor se sitúa en el lugar en donde antes estaba la línea.

Ahora escriba:

30 (pulse [INTRO])

El **+2A** busca la línea 30; como no hay ninguna con ese número, va a parar al final del programa. Por eso el cursor queda situado justamente detrás de la última línea. Si escribimos un número de línea que no existe, el **+2A** sitúa el cursor en el lugar en que la línea debería estar si realmente existiese. Ésta puede ser una forma útil de moverse en programas largos; pero tenga cuidado porque también puede ser muy peligroso. En efecto, si la línea ya existe antes de que escribamos el número y pulsemos [INTRO], ciertamente **no** existirá después.

Para listar un programa en la pantalla basta con escribir:

LIST (pulse [INTRO])

A veces se prefiere listar solamente desde cierta línea en adelante (en particular cuando se trabaja con programas largos). Para ello se escribe el número de línea adecuado detrás de la orden **LIST**.

Escriba:

LIST 15 (pulse [INTRO])

y compruebe el resultado.

Observe cómo pudimos insertar la línea 15 entre las otras dos al escribir el programa anterior. Esto habría sido imposible si sus números hubiesen sido 1 y 2, en vez de 10 y 20. Por esta razón, siempre es conveniente dejar intervalos prudenciales entre los números de línea.

(Tenga en cuenta que todos los números de línea tienen que ser enteros y estar entre 1 y 9999.)

Si en algún momento se da usted cuenta de que no ha dejado suficiente espacio entre los números de línea, puede invocar el menú de edición y renumerar el programa. Para ello pulse la tecla [EDIT] y seleccione la opción **Renumerar** del menú. El programa queda renumerado a partir de la línea 10 y con un intervalo de 10 unidades entre líneas sucesivas. Pruébelo y observe cómo cambian los números.

Ahora vamos a utilizar la orden **NEW** de BASIC. Esta orden borra el programa almacenado en la memoria del **+2A**, junto con todas sus variables. Es la orden que se debe dar al ordenador cuando se quiere empezar partiendo de cero. Escriba:

NEW

y pulse [INTRO]. De ahora en adelante ya no escribiremos 'pulse [INTRO]' cada vez que usted deba pulsar esta tecla, pero no se olvide de hacerlo.

Con el menú de presentación en la pantalla, active BASIC seleccionando la opción +3 BASIC.

A continuación transcriba cuidadosamente el siguiente programa, que convierte grados Fahrenheit en grados centígrados:

```
10 REM Conversión de temperatura
20 PRINT "Grados F","Grados C"
30 PRINT
40 INPUT "Introduzca grados F",f
50 PRINT f,
60 PRINT (f-32)*5/9
70 GO TO 40
```

Observe que, si escribe toda la línea 10 en minúsculas, BASIC sólo convierte en mayúsculas la palabra 'REM', ya que ésta es la única palabra clave que hay en la línea. Además, aunque haya escrito **GOTO** en forma de una sola palabra, el ordenador la parte en dos.

Ahora ejecute el programa. Verá que la línea 20 escribe las cabeceras en la pantalla superior. Pero ¿qué ha hecho la línea 10? Parece que el **+2A** la ha ignorado completamente; en efecto, eso es lo que ha hecho. La palabra **REM** de la línea 10 es en realidad una abreviatura de *remark* ('observación') y su único efecto es permitir que hagamos anotaciones al programa. Una sentencia **REM** consiste en la palabra **REM** seguida de cualquier cosa. El ordenador ignora todo lo que escribamos a la derecha de **REM** hasta el final de la línea.

Después de la línea 20, la 30 no hace más que escribir una línea en blanco. Cuando el **+2A** llega a la orden **INPUT** de la línea 40, espera hasta que escribamos un valor para la variable **f** (observe que el cursor está en la pantalla inferior).

Introduzca un número. El **+2A** muestra el resultado del cálculo y queda a la espera de otro número. Esto se debe a que la instrucción de la línea 70 es **GO TO 40**, que en castellano se puede leer 'ir a 40'; en otras palabras, 'en vez de salir del programa y detenerte, salta hacia atrás, a la línea 40, y continúa a partir de ella'.

Así pues, introduzca otra temperatura, luego otra, ... Quizá se pregunte usted si la máquina llegará a cansarse de hacer siempre lo mismo; pues no, no se cansa nunca. Para detener el programa haga lo siguiente: en vez de introducir otro número, pulse la tecla [A] en combinación con [SIMB]. Esto hace que aparezca la palabra **STOP**; cuando usted pulse [INTRO], el **+2A** responderá con el informe:

```
H STOP en INPUT, 40:1
```

Este informe indica que el programa se ha detenido en una instrucción **INPUT**, que es la primera (**:1**) de la línea 40.

Si desea reanudar la ejecución del programa, escriba:

```
CONTINUE
```

y el **+2A** le pedirá otro número.

Cuando se le da la orden **CONTINUE**, el **+2A** recuerda el número de línea incluido en el último informe que ha emitido (siempre que no fuera **0 OK**) y salta a esa línea, que en nuestro caso es la 40 (la de la orden **INPUT**).

Detenga nuevamente el programa y cambie la línea 70 por:

70 GO TO 31

No habrá una diferencia perceptible en la ejecución del programa, porque si el número de línea de una orden **GO TO** se refiere a una línea inexistente, el salto se efectúa hasta la siguiente línea posterior al número especificado. Análogamente ocurre con la orden **RUN** (de hecho, **RUN** equivale a **RUN 0**).

Ahora introduzca números hasta que la pantalla superior empiece a llenarse. Cuando ya esté llena, el **+2A** desplazará hacia arriba todo el contenido de la pantalla superior para hacer sitio para nuevas líneas, y el encabezamiento se saldrá de la pantalla. Este efecto se denomina *rodadura* o *desplazamiento*.

Si lo desea, ya puede detener el programa, tal y como hizo antes, y activar el editor pulsando INTRO.

Observe la sentencia **PRINT** de la línea 50. En ella la coma es muy importante.

Las comas sirven para hacer que la escritura comience en el margen izquierdo o en el centro de la pantalla (lo que quede más cerca). Así, en la línea 50 la coma hace que el ordenador escriba la temperatura centígrada a partir del centro de la línea.

Por otro lado, un punto y coma (;) hace que el siguiente número o cadena literal (grupo de caracteres) aparezca inmediatamente después del precedente.

Otro signo de puntuación que podemos utilizar en las órdenes **PRINT** es el apóstrofo ('), el cual hace que lo que se escriba a continuación aparecezca al principio de la línea siguiente. Esto mismo ocurre «por defecto» (es decir, cuando no especificamos otra cosa) al final de cada orden **PRINT**. Si desea inhibir el salto a la línea siguiente, puede poner una coma o un punto y coma al final de la sentencia **PRINT**. Para ver cómo funciona, reemplace sucesivamente la línea 50 por cada una de éstas:

50 PRINT f,
50 PRINT f;
50 PRINT f

y ejecute el programa cada vez para observar el efecto.

La línea que termina en coma lo escribe todo en dos columnas; la línea del punto y coma lo coloca todo junto; la línea que no lleva coma ni punto y coma escribe cada número en una nueva línea (esto mismo se conseguiría con **PRINT f'**).

Recuerde siempre la diferencia entre la coma y el punto y coma en las órdenes **PRINT** y no los confunda con los dos puntos (:), que son los *separadores* que debemos intercalar entre las órdenes incluidas en una misma línea; por ejemplo:

PRINT f: GO TO 40

Ahora escriba estas líneas adicionales:

100 REM Este programa recuerda nombres
110 INPUT "Escriba su nombre",n$
120 PRINT "¡Hola, ";n$;"!"
130 GO TO 110

Este programa es independiente del anterior, pero podemos guardar ambos en el **+2A** al mismo tiempo. Para ejecutar el nuevo programa dé la orden:

RUN 100

Puesto que el programa espera que introduzcamos una *cadena literal* (o sea, un carácter o un grupo de caracteres cualesquiera) en vez de un número, mostrará el cursor **L** entre comillas (**"L"**) para recordárnoslo. Así pues, escriba un nombre y pulse [INTRO].

La próxima vez volverán a aparecer las comillas, pero usted no tiene que utilizarlas si no lo desea. Por ejemplo, pruebe lo siguiente: borre las comillas pulsando [→] y dos veces [BORRAR]; después escriba:

n$

Al no haber comillas, el **+2A** sabe que tiene que realizar algún cálculo (en este caso, averiguar el valor de la variable literal **n$**, que será el nombre introducido la vez anterior). De este modo, la sentencia **INPUT** actúa como **LET n$=n$**, así que el valor de **n$** permanece inalterado.

Cuando quiera detener el programa, borre las comillas, pulse [A] en combinación con [SIMB] y finalmente pulse [INTRO].

Ahora examinemos la instrucción **RUN 100**, que provoca el salto a la línea 100 e inicia la ejecución del programa a partir de ella. Usted podría preguntarse: '¿cuál es la diferencia entre **RUN 100** y **GO TO 100**?' Pues bien, **RUN 100** empieza por borrar la pantalla y todas las variables, y después actúa igual que **GO TO 100**. En cambio, **GO TO 100** no borra nada. Hay ocasiones en que se necesita ejecutar un programa sin borrar las variables; en tales casos hay que usar **GO TO**, porque **RUN** podría ser desastrosa. Por consiguiente, no conviene que se acostumbre a escribir **RUN** sistemáticamente para poner en marcha los programas.

Otra diferencia obvia consiste en que se puede escribir **RUN** sin número de línea (y entonces la ejecución comienza a partir de la primera línea del programa), mientras que **GO TO** siempre tiene que ir seguida de un número de línea.

Tanto este programa como el de conversión de temperatura se detienen solamente cuando pulsamos [SIMB][A] en la línea de entrada. A veces se escribe por error un programa que no se detiene por sí mismo ni puede ser interrumpido por este procedimiento. Por ejemplo, escriba:

200 GO TO 200
RUN 200

Aunque la pantalla está en blanco, el programa sí está funcionando, ejecutando la línea 200 una y otra vez. Aparentemente va a continuar así para siempre, a menos que desenchufemos el cable o pulsemos el botón **REINIC**. Sin embargo, hay un remedio menos drástico: pulsar la tecla [BREAK]. El programa se detiene y el ordenador emite el informe:

L PROGR. INTERRUMPIDO, 200:1

Cada vez que termina de ejecutar una sentencia, el ordenador comprueba si está pulsada la tecla [BREAK]; si lo está, interrumpe el programa. También se puede usar esta tecla para interrumpir las operaciones con el magnetófono, la impresora y algunos otros periféricos que pueden ser conectados al **+2A**. En esos casos el informe es distinto:

D BREAK – CONT repite

En esta situación (y también en muchas otras), la instrucción **CONTINUE** repite la sentencia en la que el programa fue interrumpido y continúa directamente con la siguiente.

Ejecute otra vez el programa del 'nombre' y, cuando le pida la entrada, escriba:

n$ (después de borrar las comillas)

Puesto que **n$** es una variable que no ha sido definida, el ordenador emite el siguiente mensaje de error:

2 VARIABLE NO DEFINIDA, 110:1

Defina la variable escribiendo la orden:

LET n$="Mi nombre"

(que produce el informe **0 OK, 0:1**) y luego escriba:

CONTINUE

Compruebe que ahora ya puede usar **n$** como dato de entrada sin ningún problema.

En este caso, **CONTINUE** provoca el salto a la orden **INPUT** de la línea 110; ignora el informe producido por la sentencia **LET** porque era **OK** y salta a la orden aludida en el informe anterior, que era la 110. Esta característica puede ser muy útil, pues permite «reparar» un programa que se ha detenido a causa de un error y continuar (**CONTINUE**) a partir de donde se produjo la interrupción.

Como dijimos antes, el informe **L PROGR. INTERRUMPIDO** es especial porque, tras él, **CONTINUE** no repite la orden en la que el programa se detuvo.

Ya hemos visto las sentencias **PRINT**, **LET**, **INPUT**, **RUN**, **LIST**, **GO TO**, **CONTINUE**, **NEW** y **REM**. Todas ellas pueden ser utilizadas como órdenes directas o en líneas de programa. (Esto es válido para casi todas las órdenes de +3 BASIC; sin embargo, no es frecuente incluir **RUN**, **LIST**, **CONTINUE** y **NEW** en las líneas de programa.)

Ejercicios

1. Ponga una sentencia **LIST** en un programa, de forma que el programa se liste a sí mismo al ejecutarlo.

2. Escriba un programa que pida los precios de los artículos y escriba el importe del IVA (12 %). Utilice sentencias **PRINT** mediante las que el ordenador explique qué va a hacer y pida los precios con extravagante amabilidad. Modifique luego el programa de forma que éste pregunte el porcentaje del impuesto (con lo cual el programa valdrá también para artículos exentos de IVA y para los que estén sujetos a porcentajes diferentes).

3. Elabore un programa que escriba la suma acumulada de los números que el usuario vaya introduciendo.

4. ¿Cuál sería el efecto de **CONTINUE** y **NEW** dentro de un programa? ¿Se le ocurre alguna aplicación práctica?

Sección 3
Decisiones

Temas tratados:

CLS, IF, STOP
=, <, >, <=, >=, <>

Todos los programas que hemos visto hasta ahora eran bastante predecibles: obedecían sucesivamente las instrucciones y volvían al principio. Esto no es lo más útil que puede hacer el ordenador por nosotros; en la práctica, lo que esperamos de un ordenador es que sea capaz de tomar decisiones y obrar en consecuencia. En BASIC la instrucción que lleva a cabo la toma de decisiones tiene la siguiente forma: '**IF** (si) algo es cierto **THEN** (entonces) haz tal cosa'.

Veamos un ejemplo. Dé la orden **NEW** para borrar el programa anterior de la memoria del **+2A**, active +3 BASIC y transcriba y ejecute el siguiente programa (que, evidentemente, ha sido ideado para que jueguen dos personas):

```
10 REM Adivinar un número
20 INPUT "Introduzca un número secreto",a: CLS
30 INPUT "Adivine el número",b
40 IF b=a THEN PRINT "¡Correcto!": STOP
50 IF b<a THEN PRINT "Ese es demasiado pequeño; vuelva a intentarlo"
60 IF b>a THEN PRINT "Ese es demasiado grande; vuelva a intentarlo"
70 GO TO 30
```

Observe que la orden **CLS** (en la línea 20) significa 'borrar la pantalla' (en inglés, *clear screen*). Nosotros la hemos incluido en este programa para impedir que la otra persona vea el número secreto una vez que éste ha sido introducido.

Como puede apreciar, la sentencia **IF** tiene la forma:

IF *condición* **THEN** *xxx*

donde *xxx* representa una orden (o una secuencia de órdenes separadas por signos de dos puntos). La *condición* es algo que el ordenador tiene que evaluar y que puede dar como resultado 'verdadero' o 'falso'. Si resulta 'verdadero', las sentencias del resto de la línea (posteriores a **THEN**) son ejecutadas; de lo contrario, el ordenador las ignora y salta a la siguiente instrucción.

Las condiciones más sencillas consisten en la comparación de dos números o dos cadenas. Por ejemplo, se puede comparar dos números para averiguar si son iguales o si uno es mayor que el otro, o comparar dos cadenas para ver cuál está antes en el orden alfabético. En las comparaciones se utiliza los símbolos =, <, >, <=, >= y <>, que son los denominados *operadores de relación*:

- = se lee 'es igual a'
- < se lee 'es menor que'
- > se lee 'es mayor que'
- <= se lee 'es igual o menor que'
- >= se lee 'es igual o mayor que'
- <> se lee 'es distinto de'

En el programa que acabamos de escribir, la línea 40 compara **a** con **b**. Si son iguales, el programa es interrumpido por la orden **STOP**. El mensaje que aparece en la pantalla inferior es:

9 Sentencia STOP, 40:3

el cual indica que la tercera sentencia (es decir, la orden **STOP**) de la línea 40 hizo que el programa se detuviera.

La línea 50 averigua si **b** es menor que **a**; la línea 60, si **b** es mayor que **a**. Si una de estas condiciones es verdadera, el programa escribe el comentario apropiado y continúa en la línea 70, desde la cual salta a la 30, donde se reinicia el proceso.

Para terminar, observe que en algunas versiones de BASIC (no en +3 BASIC) la sentencia **IF** puede tener la forma:

IF *condición* **THEN** *número-de-línea*

En +3 BASIC esto equivale a:

IF *condición* **THEN GO TO** *número-de-línea*

Ejercicio

1. Pruebe el siguiente programa:

```
10 LET a=1
20 LET b=1
30 IF a>b THEN PRINT a;" es mayor"
40 IF a<b THEN PRINT b;" es mayor"
```

Antes de ejecutarlo, trate de adivinar qué va a aparecer en la pantalla.

Sección 4
Bucles

Temas tratados:

FOR, NEXT
TO, STEP

Supongamos que deseamos escribir un programa que capte cinco números por el teclado y los sume.

Un método podría ser el siguiente (no copie esto, a menos que quiera hacer prácticas de mecanografía):

```
10 LET total=0
20 INPUT a
30 LET total=total+a
40 INPUT a
50 LET total=total+a
60 INPUT a
70 LET total=total+a
80 INPUT a
90 LET total=total+a
100 INPUT a
110 LET total=total+a
120 PRINT total
```

Este método es pésimo. El programa resulta más o menos manejable cuando se trata de sumar cinco números, pero imagínese que hubiera que sumar diez (o cien).

Lo que vamos a hacer es usar una variable para contar hasta 5 y luego detener el programa (éste sí debe copiarlo):

```
10 LET total=0
20 LET contador=1
30 INPUT a
40 REM contador indica el número de veces que
   se ha captado el valor de a hasta ahora
50 LET total=total+a
60 LET contador=contador+1
70 IF contador<=5 THEN GO TO 30
80 PRINT total
```

Observe qué fácil sería modificar la línea 70 para que el programa sumara diez números, o incluso cien.

Este método es tan útil que hay dos órdenes especiales para hacerlo más fácil: la orden **FOR** y la orden **NEXT**, las cuales siempre han de ser utilizadas conjuntamente. Con ellas el programa anterior se convierte en:

```
10 LET total=0
20 FOR c=1 TO 5
30 INPUT a
40 REM c es el número de veces que se ha
   captado el valor de a hasta ahora
50 LET total=total+a
60 NEXT c
80 PRINT total
```

(Para obtener este programa a partir del anterior, basta con editar las líneas 20, 40 y 60 y borrar la línea 70.)

Observe que hemos cambiado **contador** por **c**. Esto ha sido necesario porque el nombre de la *variable de control* de un bucle **FOR...NEXT** tiene que constar de una sola letra.

El mecanismo de este programa es el siguiente: **c** va tomando sucesivamente los valores 1 (*valor inicial*), 2, 3, 4 y 5 (*límite*), y para cada uno de ellos el ordenador ejecuta las líneas 30, 40 y 50. Después, cuando **c** ya ha recorrido sus cinco valores, ejecuta la línea 80.

Ahora intente resolver el ejercicio 2 del final de esta Sección 4, que hace referencia al programa anterior.

Un refinamiento adicional de la estructura **FOR...NEXT** consiste en que la variable de control no tiene que crecer necesariamente de uno en uno, sino que podemos cambiar ese 1 por cualquier otro número sin más que incluir la cláusula **STEP** (paso) en la orden **FOR**. La forma más general de una orden **FOR** es, pues,

FOR *variable-de-control* = *valor-inicial* **TO** *límite* **STEP** *paso*

donde la *variable-de-control* es una sola letra, y el *valor-inicial*, el *límite* y el *paso* son «expresiones» (es decir, cualquier cosa que el **+2A** pueda evaluar y cuyo valor sea un número: constantes numéricas, sumas de números, variables numéricas, etc.). Pues bien, si reemplazamos la línea 20 del programa por:

```
20 FOR c=1 TO 5 STEP 3/2
```

la variable de control se verá incrementada en 3/2 cada vez que se ejecute el bucle **FOR**. Observe que podríamos haber puesto **STEP 1.5**, o haber asignado el valor del paso a una variable (por ejemplo, **p**) y luego haber especificado **STEP p**.

Con esta modificación, la variable **c** tomará los valores 1, 2.5 y 4. Observe que no hay por qué limitarse a números enteros y que, por otra parte, no es necesario que la variable

de control llegue a coincidir exactamente con el valor del límite: el bucle se reitera mientras el valor de la variable de control sea igual o menor que el del límite.

Intente resolver el ejercicio 3 del final de esta Sección 4, que hace referencia al programa anterior.

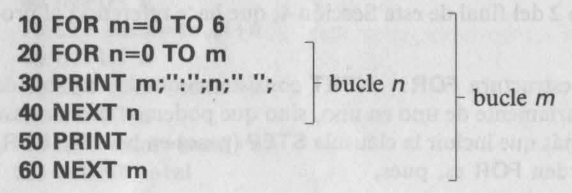
Los valores del paso pueden ser negativos en vez de positivos. Pruebe el siguiente programa, que escribe los números del 1 al 10 en orden inverso. (Acuérdese siempre de dar la orden **NEW** antes de empezar a escribir un programa nuevo.)

```
10 FOR n=10 TO 1 STEP -1
20 PRINT n
30 NEXT n
```

Dijimos antes que el programa continúa realizando bucles mientras la variable de control sea igual o menor que el límite, pero eso sólo es válido cuando no se incluye la cláusula **STEP** o cuando el valor del *paso* en ésta es positivo. Si el paso es negativo, la regla es como sigue: el bucle se repite mientras la variable de control sea igual o mayor que el límite.

Intente resolver los ejercicios 4 y 5 del final de esta Sección 4, que hacen referencia al programa anterior.

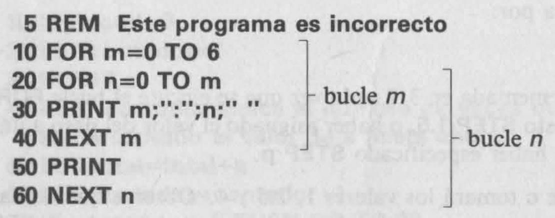
Hay que tener cuidado cuando se utiliza dos bucles **FOR...NEXT** anidados (es decir, uno dentro de otro). Pruebe el siguiente programa, que escribe todos los valores posibles de las fichas de dominó:



```
10 FOR m=0 TO 6
20 FOR n=0 TO m
30 PRINT m;":";n;" ";
40 NEXT n
50 PRINT
60 NEXT m
```


Observe que el bucle *n* está completamente dentro del bucle *m*. Esto significa que el anidamiento es correcto.

Lo que siempre se debe evitar es tener dos bucles **FOR...NEXT** solapados, como ocurre en el siguiente programa:



```
5 REM Este programa es incorrecto
10 FOR m=0 TO 6
20 FOR n=0 TO m
30 PRINT m;":";n;" "
40 NEXT m
50 PRINT
60 NEXT n
```

En resumen, si un programa contiene dos bucles **FOR...NEXT**, éstos deben estar, o bien uno dentro del otro, o bien completamente separados.

Otra cosa que hay que evitar es saltar al interior de un bucle **FOR...NEXT** desde fuera de él. BASIC sólo prepara correctamente la variable de control cuando ejecuta su sentencia **FOR**; si el ordenador no ejecuta una sentencia **FOR**, la sentencia **NEXT** lo dejará completamente confundido. La consecuencia más probable es el mensaje de error **'NEXT sin FOR'**, o bien **'VARIABLE NO DEFINIDA'**.

Nada impide utilizar un bucle **FOR...NEXT** en una orden directa. Por ejemplo, pruebe la siguiente:

```
FOR m=0 TO 10: PRINT m: NEXT m
```

Hay una forma (un tanto artificial) de repetir automáticamente una orden directa. (Recuerde que **GO TO** necesita un número de línea, y eso es lo que una orden directa no puede tener.) Por ejemplo,

```
FOR m=0 TO 1 STEP 0: INPUT a: PRINT a: NEXT m
```

El hecho de que el paso sea 0 hace que la orden se repita indefinidamente.

Pero este método no es muy recomendable: si se produce un error se pierde la orden y hay que volver a escribirla; además, **CONTINUE** no funcionará en este caso.

Ejercicios

1. Asegúrese de que entiende perfectamente que una variable de control no sólo tiene un nombre y un valor, como una variable ordinaria, sino además un límite, un paso y una conexión con la sentencia **FOR** correspondiente. Por otra parte, cuando se ejecuta la sentencia **FOR**, toda esta información se encuentra disponible y es suficiente para que la sentencia **NEXT** sepa cómo tiene que modificar el valor de la variable, si debe o no repetir el bucle y, de ser así, a qué línea tiene que saltar.

2. Ejecute el tercer programa de esta sección y luego dé la orden:

   ```
   PRINT c
   ```

   ¿Por qué la respuesta es 6 y no 5?

   (*Respuesta*: la orden **NEXT** de la línea 60 es ejecutada cinco veces, y en cada una de ellas suma 1 a **c**. La última vez **c** se convierte en 6; por eso la orden **NEXT** decide no repetir el bucle, sino continuar, ya que **c** ha sobrepasado el límite).

3. ¿Qué ocurre si ponemos **STEP 2** al final de la línea 20 del tercer programa? Pruebe con **STEP 10**.

   Ahora modifique el tercer programa de forma que, en lugar de sumar automáticamente los cinco números, pregunte al usuario cuántos quiere sumar. Al ejecutar este progra-

ma, ¿qué ocurriría si usted respondiese con un 0 (lo que significaría que no quiere sumar ningún número)? ¿Cree que esto le causaría algún problema al **+2A**, aun estando claro lo que usted quiere decir?

4. En la línea 10 del cuarto programa de esta sección, cambie **10** por **100** y ejecute el programa. En la pantalla aparecerán los números del 100 al 79 y luego el mensaje **¿MAS?**. El **+2A** le da así la oportunidad de observar los números antes de que sean desplazados hacia arriba. Si pulsa [N], [BREAK] o la barra espaciadora, el programa se detendrá con el mensaje **D BREAK – CONT repite**. Si pulsa cualquier otra tecla, el programa escribirá otras 22 líneas y volverá a preguntarle si desea otra 'rodadura'.

5. Borre la línea 30 del cuarto programa. Cuando ejecute la nueva versión, el programa escribirá el primer número y se detendrá con el mensaje **0 OK**. Si a continuación escribe:

   NEXT n

   el programa ejecutará una vez el bucle y escribirá el número siguiente.

Sección 5
Subrutinas

Temas tratados:

GO SUB, RETURN

A veces nos encontramos con que diferentes partes del programa tienen que realizar funciones bastante similares, y esto fácilmente puede llevarnos a escribir las mismas instrucciones varias veces con diferente número de línea. Afortunadamente, en BASIC disponemos de un recurso que nos permite evitar este despilfarro: podemos escribir las instrucciones una sola vez, en lo que denominamos una *subrutina*, y luego invocar la subrutina siempre que sea necesario.

El control de las subrutinas se realiza con dos sentencias: **GO SUB** y **RETURN**. Para invocar una subrutina se da una orden del tipo:

GO SUB *xxx*

donde *xxx* es el número de la primera línea de la subrutina. El mecanismo es similar al de **GO TO** *xxx*, con la diferencia de que el ordenador recuerda dónde estaba la sentencia **GO SUB** y así sabe a dónde tiene que volver cuando termine de ejecutar la subrutina.

(Por si le interesa saberlo, el **+2A** «recuerda» en qué punto del programa estaba la sentencia **GO SUB** porque almacena la *dirección de retorno* en la *pila* de **GO SUB**).

Cuando el ordenador encuentra la orden

RETURN

sabe que ahí termina la subrutina; entonces recuerda dónde estaba la sentencia **GO SUB** (tomando de la pila la dirección de retorno) y continúa a partir de la sentencia siguiente.

Como ejemplo, echemos un vistazo de nuevo al programa de adivinar números. Reescríbalo de la siguiente forma:

```
 10 REM Programa de adivinar números, reorganizado
 20 INPUT "Introduzca un número secreto",a: CLS
 30 INPUT "Adivine el número",b
 40 IF b=a THEN PRINT "¡Correcto!": STOP
 50 IF b<a THEN GO SUB 100
 60 IF b>a THEN GO SUB 100
 70 GO TO 30
100 PRINT "Vuelva a intentarlo"
110 RETURN
```

La sentencia **GO TO 30** de la línea 70, y la sentencia **STOP** del programa siguiente, son muy importantes; si no fuera por ellas, los programas entrarían en las subrutinas y se produciría un error (**'7 RETURN sin GO SUB'**) al llegar a la sentencia **RETURN**.

El siguiente programa utiliza una subrutina (líneas 100 a 150) que escribe la tabla de multiplicar correspondiente al *parámetro* **n**. La orden **GO SUB 100** que invoca la subrutina puede estar en cualquier lugar del programa. Cuando BASIC encuentra la orden **RETURN** en la línea 150 de la subrutina, el control retorna al programa principal, que continúa funcionando a partir de la sentencia siguiente a aquélla en que se encontraba el **GO SUB** original. Al igual que ocurría con **GO TO**, se puede escribir **GO SUB** en una sola palabra: **GOSUB**.

```
10 REM Tabla de multiplicar del 2, el 5, el 10 y el 11
20 LET n=2: GO SUB 100
30 LET n=5: GO SUB 100
40 LET n=10: GO SUB 100
50 LET n=11: GO SUB 100
60 STOP
70 REM Fin del programa principal, comienzo de la subrutina
100 PRINT "Tabla de multiplicar por ";n
110 FOR v=1 TO 9
120 PRINT v;" x ";n;" = ";v*n
130 NEXT v
140 PRINT
150 RETURN
```

Una subrutina puede invocar a otra, o incluso a sí misma. (Una rutina que se invoca a sí misma es una *rutina recursiva*).

Sección 6
Datos en los programas

Temas tratados:

READ, DATA, RESTORE

En algunos de los programas anteriores hemos visto que la información (es decir, los datos) puede ser captada por el programa mediante la sentencia **INPUT**. En ocasiones esto resulta muy tedioso, especialmente cuando buena parte de la información que se requiere es la misma en todas las ejecuciones del programa. Podemos ahorrar un tiempo considerable utilizando las sentencias **READ**, **DATA** y **RESTORE**. Veamos un ejemplo:

```
10 READ a,b,c
20 PRINT a,b,c
30 DATA 1,2,3
```

Una sentencia **READ** consiste en la palabra clave **READ** seguida de una lista de nombres de variables separados entre sí por comas. Funciona de modo bastante parecido a **INPUT**, con la ventaja de que, en vez de hacernos escribir los valores que hay que asignar a las variables, el ordenador los lee en la sentencia **DATA**.

Cada sentencia **DATA** es una lista de expresiones (numéricas o literales) separadas por comas. Las sentencias **DATA** pueden estar en cualquier lugar del programa, ya que el **+2A** las ignora completamente, excepto cuando está ejecutando una sentencia **READ**. Podemos imaginarnos las expresiones de todas las sentencias **DATA** del programa como si estuvieran colocadas todas juntas, formando una larga lista: la *lista de datos*. La primera vez que el **+2A** lee (con **READ**) un valor, consulta la primera expresión de la lista de datos; la siguiente vez, lee la segunda; y así sucesivamente, según va encontrando instrucciones **READ**, sigue su camino a lo largo de la lista de datos. (Si trata de leer más allá del fin de la lista, se produce un error).

Observe que es una pérdida de tiempo poner instrucciones **DATA** en una orden directa, ya que **READ** no las encontrará. Las sentencias **DATA** tienen que estar en líneas de programa.

Vamos a ver cómo funciona todo esto en el programa que acabamos de transcribir. La línea 10 le pide al ordenador que lea tres datos y los asigne a las variables **a**, **b** y **c**. La línea 20 le pide que escriba (**PRINT**) los valores de esas variables. La sentencia **DATA** de la línea 30 proporciona los valores de **a**, **b** y **c** para que los lea la línea 10.

La información contenida en una sentencia **DATA** puede ser leída por un bucle **FOR...NEXT**. Escriba lo siguiente:

```
10 FOR n=1 TO 6
20 DATA 2,4,6,8,10,12
30 READ d
40 PRINT d
50 NEXT n
```

Los dos programas anteriores han demostrado que las sentencias **DATA** pueden estar en cualquier lugar (antes o después de la instrucción **READ**).

Al ejecutar este último programa, la sentencia **READ** avanza por la lista de datos en cada pasada por el bucle **FOR...NEXT**.

Las sentencias **READ** pueden también asignar valores a variables literales. Por ejemplo,

```
10 FOR a=1 TO 7
20 READ n$
30 PRINT n$
40 DATA "a","ante","bajo","cabe","con","contra","de"
50 NEXT a
```

No siempre hay que leer las sentencias **DATA** en orden, de la primera a la última; de hecho, podemos saltar de una sentencia **DATA** a otra mediante la orden **RESTORE**. La forma de esta orden es:

RESTORE *xxx*

donde *xxx* es el número de la línea en que se encuentra la sentencia **DATA** que debe ser leída. La orden **RESTORE** sin número de línea hace que el *puntero de datos* apunte hacia la primera sentencia **DATA** del programa.

Transcriba y pruebe el siguiente programa:

```
 10 DATA 1,2,3,4,5
 20 DATA 6,7,8,9
 30 GO SUB 110
 40 GO SUB 110
 50 GO SUB 110
 60 RESTORE 20
 70 GO SUB 110
 80 RESTORE
 90 GO SUB 110
100 STOP
110 READ a,b,c
120 PRINT a'b'c
130 PRINT
140 RETURN
```

La orden **GO SUB 110** invoca una subrutina que lee los tres datos siguientes y después los escribe (**PRINT**). Observe cómo la orden **RESTORE** decide qué elementos son leídos en cada caso.

Borre la línea 60 y ejecute este programa de nuevo para ver qué ocurre.

Sección 7
Expresiones

Temas tratados:

Operaciones: +, −, *, /

Expresiones, notación científica, nombres de variables

Ya hemos visto algunas de las formas en las que el **+2A** realiza cálculos con números. Además de las cuatro operaciones aritméticas, +, −, * y / (recuerde que en BASIC * es el símbolo de la multiplicación y / el de la división), sabe encontrar el valor de una variable, dado su nombre.

La sentencia

```
LET impuesto=suma*15/100
```

es un ejemplo de cómo se puede combinar los cálculos. Una combinación tal como **suma*15/100** es lo que se llama *expresión*. Por consiguiente, una expresión es una forma abreviada de decirle al **+2A** cómo debe realizar varios cálculos, uno a continuación de otro. En nuestro ejemplo, la expresión **suma*15/100** significa: 'busca el valor de la variable llamada **suma**, multiplícalo por 15 y divídelo por 100'.

En las expresiones que contienen *, /, + o −, la multiplicación y la división son prioritarias con respecto a la suma y la resta (es decir, el ordenador las realiza antes que la suma y la resta). La multiplicación y la división tienen la misma prioridad una que la otra, lo que significa que el **+2A** las realiza en el orden en que las encuentra en la expresión (de izquierda a derecha). Las siguientes operaciones que realiza el **+2A** son la suma y la resta; como también tienen el mismo nivel de prioridad, el ordenador las calcula de izquierda a derecha.

Así pues, en la expresión **8−12/4+2*2** la primera operación que se efectúa es la división **12/4**, cuyo resultado es **3**, por lo que la expresión equivale a **8−3+2*2**.

La siguiente operación que se lleva a cabo es la multiplicación **2*2**, que da **4**, así que la expresión se convierte entonces en **8−3+4**.

El siguiente paso es restar **8−3**, que da **5**; la expresión pasa a ser **5+4**. Finalmente, se realiza la suma y el resultado es **9**.

Para comprobarlo, dé la orden

```
PRINT 8-12/4+2*2
```

En la Sección 31 de este capítulo daremos la lista completa de las prioridades de las operaciones matemáticas (y lógicas).

Se puede cambiar la prioridad de los cálculos dentro de una expresión mediante el uso adecuado de los paréntesis. Los cálculos que van entre paréntesis son realizados antes que todos los demás; por lo tanto, si en la expresión anterior queremos que se empiece por calcular la suma **4+2**, basta con escribirla entre paréntesis. Compruébelo con la orden:

```
PRINT 8-12/(4+2)*2
```

El resultado es ahora **4** en lugar de **9**.

Las expresiones son útiles por el hecho de que, siempre que el **+2A** esté esperando un número, nosotros podemos darle una expresión en su lugar y él hallará la respuesta.

También podemos «sumar» cadenas (o variables literales) en una expresión. Por ejemplo,

```
10 LET a$="arroz"
20 LET b$=" con "
30 LET c$="leche"
40 LET d$=a$+b$+c$
50 PRINT d$
```

Ha llegado el momento de que digamos qué es lo que se puede y lo que no se puede usar como nombre de variable. Como ya sabemos, el nombre de una variable literal tiene que ser una sola letra seguida de **$**, y el nombre de la variable de control de los bucles **FOR...NEXT** también tiene que consistir en una sola letra. Pero los nombres de las variables numéricas ordinarias son mucho más libres; pueden constar de dígitos o letras cualesquiera, siempre que el primer carácter sea una letra. Incluso podemos introducir espacios en los nombres de las variables, para hacerlos más legibles, aunque el **+2A** los ignorará a todos los efectos, salvo en los listados. Además, es indiferente escribir los nombres en mayúsculas o en minúsculas. No obstante, hay algunas restricciones acerca de los nombres de variables: no pueden coincidir con las palabras clave y, en general, si un nombre de variable contiene una palabra clave con espacios a los lados, BASIC no lo aceptará.

He aquí unos cuantos ejemplos de nombres de variable válidos:

```
x
cualquier cosa
t42
este nombre no es conveniente porque es demasiado largo
tobeornottobe
espacios y mayusculas mezclados
EspaciosyMayusculasMezclados
```

(Observe que los dos últimos nombres son considerados iguales y se refieren a la misma variable.)

Los siguientes nombres de variable no son válidos:

pi	(**PI** es una palabra clave)
2001	(sólo contiene dígitos)
42t	(comienza con dígito)
I Love New York	(contiene **NEW** entre espacios)
to be or not to be	(**TO**, **OR** y **NOT** son palabras clave de BASIC)
3 osos	(comienza con un dígito)
M*A*S*H	(* no es letra ni dígito)
M−30	(– no es letra ni dígito)

Los valores numéricos pueden ser representados por un número y un exponente (en *notación científica*). Pruebe las siguientes órdenes:

```
PRINT 2.34e0
PRINT 2.34e1
PRINT 2.34e2
```

y así sucesivamente hasta

```
PRINT 2.34e15
```

PRINT sólo puede escribir los números con ocho cifras significativas. Pruebe la siguiente orden:

```
PRINT 4294967295, 4294967295−429e7
```

Esto demuestra que el ordenador guarda los dígitos de 4294967295, aunque no es capaz de mostrarlos todos de una vez.

El **+2A** trabaja en *aritmética de punto flotante*, lo que significa que guarda por separado los dígitos del número (la *mantisa*) y la posición del punto decimal (el *exponente*). Este método no siempre da resultados exactos, ni siquiera con números enteros. Escriba:

```
PRINT 1e10+1−1e10,1e10−1e10+1
```

+3 BASIC almacena los números con unos nueve dígitos y medio de precisión, de modo que **1e10** es demasiado grande como para ser almacenado con precisión absoluta. La inexactitud (en este caso cerca de 2) es mayor que 1, y por tanto los números 1e10 y 1e10+1 le parecen iguales al ordenador.

Un ejemplo aún más peculiar es:

```
PRINT 5e9+1−5e9
```

Aquí la inexactitud de 5e9 es sólo de alrededor de 1, y el 1 que debe ser sumado se redondea hasta 2. Los números 5e9+1 y 5e9+2 le parecen iguales al ordenador.

El número entero más grande que se puede almacenar con completa exactitud es 4294967294.

La cadena "", que no tiene caracteres, se llama *cadena vacía* o *cadena nula*. Recuerde que los espacios son significativos y que una cadena vacía no es lo mismo que una que sólo contenga espacios.

Pruebe

```
PRINT "¿Leiste "El Pais" ayer?"
```

Al pulsar INTRO aparece el cursor parpadeante en rojo, que, como sabemos, indica la presencia de un error en algún lugar de la línea. Cuando el **+2A** encuentra las comillas al principio de **"El Pais"**, da por supuesto que marcan el final de la cadena **"¿Leiste "**, y entonces no sabe qué significa **El Pais**.

Hay un recurso especial para evitar esto: cuando necesite que las comillas formen parte de una cadena, escríbalas repetidas. Por ejemplo,

```
PRINT "¿Leiste ""El Pais"" ayer?"
```

Como puede ver en la pantalla, las comillas sólo aparecen una vez (pero usted tiene que escribirlas dos veces para que sean reconocidas).

Sección 8
Cadenas literales

Temas tratados:

Disección de cadenas, **TO**

Dada una cadena, una *subcadena* de ella consiste en varios de sus caracteres tomados secuencialmente. Así, **"cadena"** es una subcadena de **"cadena mayor"**, pero **"c mayor"** y **"caneda"** no lo son.

Existe una notación específica para describir subcadenas que puede ser aplicada a expresiones literales cualesquiera. Su forma general es:

expresión-literal (*principio* **TO** *fin*)

Así, por ejemplo

"abcdef"(2 TO 5)

es igual a **bcde**.

Si se omite el *principio*, BASIC toma por defecto el 1; si se omite el *fin*, toma la longitud total de la cadena. De este modo,

"abcdef"(TO 5) es igual a **abcde**
"abcdef"(2 TO) es igual a **bcdef**
"abcdef"(TO) es igual a **abcdef**

Esta última expresión se puede escribir también en la forma **"abcdef"()**.

Hay una forma ligeramente diferente en la que se omite el **TO** y sólo se escribe un número:

"abcdef"(3) equivale a **"abcdef"(3 TO 3)** que es igual a **c**

Aunque normalmente el *principio* y el *fin* deben hacer referencia a caracteres que realmente existan en la cadena, esta regla está supeditada a otra: si el *principio* es mayor que el *fin*, el resultado es la cadena vacía. Así,

"abcdef"(5 TO 7)

da el error **'3 SUBINDICE INCORRECTO'**, porque la cadena sólo tiene 6 caracteres y el valor 7 es, por consiguiente, demasiado grande; pero

"abcdef"(8 TO 7) y **"abcdef"(1 TO 0)**

son iguales a la cadena vacía y, por lo tanto, están permitidas.

El *principio* y el *fin* no deben ser negativos; si lo son se produce el error **'B ENTERO EXCEDE MARGEN'**. El siguiente programa ilustra algunas de estas reglas:

```
10 LET a$="abcdef"
20 FOR n=1 TO 6
30 PRINT a$(n TO 6)
40 NEXT n
```

Escriba **NEW** cuando haya probado este programa y luego transcriba el siguiente:

```
10 LET a$="1234567890"
20 FOR n=1 TO 10
30 PRINT a$(n TO 10),a$((11-n) TO 10)
40 NEXT n
```

Si la cadena no es constante, sino variable, también podemos asignar valores a sus subcadenas. Por ejemplo, escriba:

```
LET a$="Me gusta mi Sinclair"
```

y después

```
LET a$(13 TO 20)="Amstrad*****"
```

Ahora dé la orden

```
PRINT a$
```

Observe que la subcadena **a$(13 TO 20)** sólo tiene ocho caracteres y que por eso en la segunda sentencia de asignación sólo se utiliza los ocho primeros caracteres de **Amstrad *******. Los cuatro restantes, ********, son ignorados. Ésta es una característica de la asignación de valores a subcadenas: la subcadena debe tener exactamente la misma longitud antes que después de la asignación. Para ello, si la cadena asignada es demasiado larga, BASIC la recorta por la derecha; si es demasiado corta, la completa con espacios por la derecha. Esto es algo parecido a lo que hacía Procrustes, un posadero que siempre adaptaba los huéspedes al tamaño de la cama: o los estiraba en el potro o les cortaba los pies.
Las expresiones literales complicadas tienen que estar entre paréntesis para poder extraer subcadenas de ellas. Por ejemplo,

"abc"+"def"(1 TO 2) es igual a **"abcde"**
("abc"+"def")(1 TO 2) es igual a **"ab"**

Ejercicio

1. Diseñe un programa que escriba el día de la semana recurriendo a la extracción de subcadenas. (*Sugerencia.* Forme la cadena con **LunMarMieJueVieSabDom.**)

Sección 9
Funciones

Temas tratados:

LEN, STR$, VAL, SGN, ABS, INT, SQR
DEF, FN

Para explicar en qué consisten las funciones, tomemos como ejemplo la máquina de hacer zumo. Usted introduce una pieza de fruta por un extremo, pone en marcha la máquina y por el otro lado sale el zumo. De una naranja sale zumo de naranja; de una pera, zumo de pera; y de una manzana, zumo de manzana.

Las funciones son como las máquinas de hacer zumo, aunque hay una diferencia: trabajan con números y cadenas en vez de con frutas. Usted suministra un valor (el *argumento*), la función lo *procesa* realizando con él algunos cálculos y, finalmente, entrega otro valor: el *resultado*.

Entrada de fruta → Máquina de hacer zumo → Zumo
Argumento → Función → Resultado

Diferentes argumentos producen diferentes resultados; si el argumento es completamente inadecuado, la función no puede procesarlo y BASIC emite un mensaje de error.

Al igual que en la industria puede haber diferentes máquinas para fabricar diferentes productos (una para zumos, otra para manteles, una tercera para guantes de goma, etc.), en el ordenador necesitamos distintas funciones para realizar los diferentes cálculos. Cada función tiene un nombre que la distingue de todas las demás.

Para utilizar una función se escribe su nombre seguido del argumento; BASIC calcula la función cuando evalúa la expresión en la que aquélla interviene.

Por ejemplo, hay una función llamada **LEN** que da como resultado la longitud de una cadena. El argumento de **LEN** es la cadena cuya longitud se desea hallar. Por ejemplo, si damos la orden

```
PRINT LEN "Spectrum +2A"
```

el ordenador responde con el número 11, es decir, el número de caracteres (incluidos los espacios) de que consta la cadena **Spectrum +2A**.

Si mezclamos funciones y operaciones en una sola expresión, aquéllas son evaluadas antes que éstas. Pero, por otra parte, podemos alterar este orden utilizando paréntesis. Por ejemplo, las dos expresiones siguientes sólo se distinguen en los paréntesis y, precisamente por ello, los cálculos son realizados en un orden completamente distinto en cada caso (aunque los resultados finales son idénticos):

LEN "Pepe " + LEN "y Manolo"	**LEN ("Pepe " + "y Manolo")**
↓	↓
5 + LEN "y Manolo"	**LEN ("Pepe y Manolo")**
↓	↓
5 + 8	**LEN "Pepe y Manolo"**
↓	↓
13	**13**

Veamos unas cuantas funciones más:

STR$ convierte números en cadenas. Su argumento es un número; su resultado es la cadena que aparecería en la pantalla si el número hubiera sido escrito por **PRINT**. Observe que el nombre de la función termina en un signo **$** para indicar que el resultado es una cadena. Por ejemplo,

LET a$=STR$ 1e2

produce exactamente el mismo efecto que

LET a$="100"

Otro ejemplo:

PRINT LEN STR$ 100.0000

da como resultado **3**, ya que **STR$ 100.0000** es igual a la cadena **100**, cuya longitud es **3** caracteres.

VAL es la función inversa de **STR$**: convierte cadenas en números. Por ejemplo,

VAL "3.5"

produce el número **3.5**.

Decimos que **VAL** es la función inversa de **STR$** porque si tomamos un número cualquiera, le aplicamos **STR$** y después **VAL**, el resultado final es el número original.

En cambio, si tomamos una cadena, le aplicamos **VAL** y luego **STR$**, no siempre obtendremos la cadena original.

VAL es una función muy potente, ya que la cadena que le entregamos como argumento no necesariamente ha de tener la forma de un número constante, sino que puede tener la de una expresión numérica. Así, por ejemplo,

VAL "2*3"

da como resultado el número **6**; incluso

VAL ("2"+"*3")

produce el número **6**. En este último caso han intervenido dos procesos. En primer lugar se ha producido la *concatenación* de **"2"** y **"*3"** para dar la cadena **"2*3"**; después **VAL** ha suprimido las comillas y ha calculado el valor del producto **2*3**, que es **6**.

Existe otra función, bastante similar a **VAL**, aunque probablemente menos útil, llamada **VAL$**. Su argumento sigue siendo una cadena, pero su resultado también lo es. Para ver cómo funciona, recuerde la forma en que **VAL** realiza los cálculos: primero evalúa su argumento hasta obtener una cadena sencilla; después suprime las comillas y todo lo que queda lo evalúa como número. En el caso de **VAL$**, el primer paso es el mismo, pero, una vez eliminadas las comillas en el segundo paso, todo lo que queda se evalúa como cadena. Por ejemplo,

VAL$ """Ursula""" es igual a **"Ursula"**

Haga ahora

LET a$="99"

y pida al ordenador que escriba todo lo siguiente: **VAL a$**, **VAL "a$"**, **VAL """a$"""**, **VAL$ a$**, **VAL$ "a$"** y **VAL$ """a$"""**. Algunas de ellas funcionarán y otras no; intente explicar todas las respuestas.

SGN es la función 'signo'. Es la primera función que nos encontramos que no tiene nada que ver con las cadenas, ya que tanto su argumento como su resultado son números. El resultado es +1 si el argumento es positivo, 0 si el argumento es 0, y −1 si el argumento es negativo.

ABS es otra función cuyos argumento y resultado son números. Convierte el argumento en un número positivo (que es el resultado) olvidando el signo. Así, por ejemplo,

ABS −3.2

es igual a

ABS 3.2

que, sencillamente, es igual a **3.2**.

INT significa 'parte entera', un número entero, posiblemente negativo. Esta función convierte un número fraccionario (con decimales) en un entero desechando los decimales. Así, por ejemplo,

INT 3.9

da **3**.

Tenga cuidado al aplicar esta función a números negativos, ya que siempre los redondea hacia abajo. Por ejemplo,

INT −3.1

es igual a **−4**.

SQR calcula la raíz cuadrada del argumento; es decir, da un resultado que, multiplicado por sí mismo, es el argumento. Por ejemplo,

SQR 4

es igual a **2** porque 2*2=4.

SQR 0.25

es igual a **0.5** porque 0.5*0.5=0.25.

SQR 2

es (aproximadamente) igual a **1.4142136** porque 1.4142136*1.4142136≃2.

Si multiplicamos cualquier número (incluso uno negativo) por sí mismo, el producto siempre es positivo. Esto significa que los números negativos no tienen raíz cuadrada; por consiguiente, si aplicamos **SQR** a un argumento negativo, obtenemos el mensaje de error **'A ARG. INVALIDO'**.

Aparte de las funciones que BASIC nos ofrece, podemos también definir otras. Sus nombres consistirán en las letras **FN** seguidas de una letra de nuestra elección (si el resultado va a ser un número) o en **FN** seguida de otra letra y de **$** (si el resultado va a ser una cadena). Estas funciones son mucho más estrictas en lo que se refiere a los paréntesis (el argumento siempre tiene que figurar entre paréntesis).

Una función se define poniendo una sentencia **DEF** en algún lugar del programa. Por ejemplo, la siguiente sentencia define la función **FN c**, cuyo resultado es el cuadrado del argumento:

10 DEF FN c(x)=x*x: REM el cuadrado de x

La letra **c** que sigue a **DEF FN** es el nombre que hemos elegido para la función. La **x** entre paréntesis es el nombre por el cual nos referiremos al argumento de la función. Para esto se puede utilizar una letra cualquiera (pero sólo una), o bien, si el argumento es una cadena, una sola letra seguida de **$**.

Tras el signo = viene la verdadera definición de la función. Ésta puede ser cualquier expresión, y puede hacer referencia al argumento utilizando el nombre que le hemos dado (en este caso, **x**), como si fuera cualquier otra variable.

Una vez ejecutada la sentencia **DEF**, podemos usar la función de la misma manera que las intrínsecas de BASIC: escribiendo su nombre, **FN c**, seguido por el argumento entre paréntesis. (Recuerde siempre que en las funciones definidas por el usuario el argumento tiene que estar entre paréntesis.) Practique unas cuantas veces:

```
PRINT FN c(2)
PRINT FN c(3+4)
PRINT 1 + INT FN c (LEN "pollo"/2+3)
```

Decíamos antes que **INT** siempre redondea hacia abajo. Para redondear al entero más cercano, sume 0.5 al número antes de aplicarle **INT**; si no quiere tener que hacer esto cada vez que necesite redondear un número, puede encomendarle la tarea a una función creada al efecto:

```
20 DEF FN r(x) = INT (x+0.5): REM da x redondeado al entero más cercano
```

Después puede comprobar que, por ejemplo,

FN r(2.9)	es igual a	**3**	**FN r(2.4)**	es igual a	**2**
FN r(−2.9)	es igual a	**−3**	**FN r(−2.4)**	es igual a	**−2**

Compare estas respuestas con las que se obtiene empleando **INT** en lugar de **FN r**. Transcriba y ejecute este programa:

```
10 LET x=0: LET y=0: LET a=10
20 DEF FN p(x,y)=a+x*y
30 DEF FN q()=a+x*y
40 PRINT FN p(2,3),FN q()
```

En este programa hay muchas sutilezas. Primero, las funciones pueden tener varios argumentos, e incluso no tener ninguno (lo que no se puede omitir es los paréntesis).

Segundo, las sentencias **DEF** pueden estar en cualquier lugar del programa. Cuando el ordenador ha ejecutado la línea 10, se salta las líneas 20 y 30 para llegar a la 40. Sin embargo, tienen que estar en alguna parte del programa, no en una orden directa.

Tercero, tanto **x** como **y** son nombres de variables en el conjunto del programa y además son los nombres de los argumentos de la función **FN p**. Ésta olvida temporalmente las variables llamadas **x** e **y**, pero, puesto que no tiene ningún *argumento* con el nombre de **a**, sigue recordando la *variable* **a**. De este modo, cuando **FN p(2,3)** está siendo evaluada, **a** tiene el valor 10 porque es la variable, **x** tiene el valor 2 porque es el primer argumento, e **y** tiene el valor 3 porque es el segundo argumento. Así que el resultado es 10+2*3, que es igual a 16. Por otra parte, cuando **FN q()** está siendo evaluada, no hay argumentos, de modo que **a**, **x** e **y** siguen refiriéndose a las variables y, por lo tanto, tienen los valores 10, 0 y 0, respectivamente. La respuesta en este caso es 10+0*0, que es igual a 10.

Cambie ahora la línea 20 por:

20 DEF FN p(x,y)=FN q()

Esta vez **FN p(2,3)** tendrá el valor 10, ya que **FN q** volverá a las variables **x** e **y** y no utilizará los argumentos de **FN p**.

Algunas versiones de BASIC (aunque no +3 BASIC) tienen las funciones **LEFT$**, **RIGHT$**, **MID$** y **TL$**.

- **LEFT$(*a$*,*n*)** da la subcadena de *a$* consistente en los *n* primeros caracteres.
- **RIGHT$(*a$*,*n*)** da la subcadena de *a$* consistente en los últimos caracteres, a partir del *n*-ésimo.
- **MID$(*a$*,*n1*,*n2*)** da la subcadena de *a$* consistente en los *n2* caracteres tomados a partir del *n1*-ésimo.
- **TL$(*a$*)** da la subcadena de *a$* consistente en todos sus caracteres menos el primero.

Podemos definir funciones de usuario que produzcan los mismos resultados:

```
10 DEF FN t$(a$)=a$(2 TO ): REM TL$
20 DEF FN l$(a$,n)=a$( TO n): REM LEFT$
```

Compruebe que estas funciones operan con cadenas de longitud 0 y 1. Observe que nuestra **FN l$** tiene dos argumentos; uno es un número y el otro una cadena. Una función puede tener hasta 26 argumentos numéricos (número de letras del alfabeto, excluida la 'ñ') y, al mismo tiempo, hasta 26 argumentos literales.

Ejercicio

1. Utilice la función **FN c(x)=x*x** para probar **SQR**. Observará que

   FN c(SQR x)

   es igual a **x** si se pone cualquier número positivo en lugar de **x**, y que

   SQR FN c(x)

   es igual a **ABS x** tanto si **x** es positivo como si es negativo. (¿Por qué aparece **ABS** aquí?)

Sección 10
Funciones matemáticas

Temas tratados:

↑

PI, EXP, LN, SIN, COS, TAN, ASN, ACS, ATN

En esta sección vamos a ocuparnos de las funciones matemáticas que puede manejar el **+2A**. Es posible que usted nunca haya utilizado ninguna de estas funciones; si se le hace pesado continuar, no le importe saltarse esta sección. Examinaremos la operación ↑ (elevar a una potencia), las funciones **EXP** y **LN** y las funciones trigonométricas **SIN**, **COS**, **TAN**, y sus inversas, **ASN**, **ACS** y **ATN**.

↑ y EXP

Elevar un número a una potencia es multiplicar el número por sí mismo cierto número de veces. Esta operación se indica normalmente escribiendo el segundo número (el *exponente*) a la derecha del primero y un poco más arriba; obviamente, esto sería difícil en un ordenador, por lo que utilizamos el símbolo ↑. Por ejemplo, las potencias de 2 son:

2↑1=2
2↑2=2*2=4 (2 al cuadrado, escrito normalmente 2^2)
2↑3=2*2*2=8 (2 al cubo, escrito normalmente 2^3)
2↑4=2*2*2*2=16 (2 a la cuarta, escrito normalmente 2^4)
etcétera.

Así, al nivel más elemental, *a*↑*b* significa '*a* multiplicado por sí mismo *b* veces'; pero, evidentemente, esto sólo tiene sentido si *b* es un número entero positivo. Para encontrar una definición que sirva para otros valores de *b*, consideremos la regla:

a↑(*b*+*c*) es igual a *a*↑*b***a*↑*c*

(Observe que ↑ tiene una prioridad mayor que * y /, de forma que, cuando hay varias operaciones en una expresión, las potencias son evaluadas antes que los productos y las divisiones.) Cualquiera puede aceptar sin demasiados reparos que esta regla es válida cuando *a* y *b* son números enteros positivos; si queremos que funcione también cuando no lo sean, nos vemos forzados a aceptar que

a↑0 es igual a 1
a↑(−*b*) es igual a 1/*a*↑*b*
a↑(1/*b*) es igual a la *b*-ésima raíz de *a*, es decir, el número que hay que multiplicar por sí mismo *b* veces para obtener *a*

y que

$a\uparrow(b*c)$ es igual a $(a\uparrow b)\uparrow c$

Si usted no ha visto nunca antes nada de esto, no intente aprendérselo de memoria a la primera; basta con que recuerde que:

$a\uparrow(-1)$ es igual a $1/a$

y que

$a\uparrow(1/2)$ es igual a **SQR** a

Puede ser, cuando se familiarice con estas fórmulas, que todas las demás empiecen a cobrar sentido.

Experimente probando este programa:

```
10 INPUT a,b,c
20 PRINT a↑(b+c),a↑b*a↑c
30 GO TO 10
```

Por supuesto, si la regla que dimos antes es cierta, en cada pasada por la línea 20 los dos números que escriba el **+2A** serán iguales. (Observe que, debido al modo en que el ordenador calcula las potencias, el número que está a la izquierda de ↑, **a** en este caso, nunca debe ser negativo.)

Un ejemplo bastante típico de aplicación de esta operación es el interés compuesto. Imagine que invierte una parte de su dinero en una sociedad inmobiliaria que le produce un 15% de interés anual. Después de un año no tendrá solamente el 100% de lo invertido, sino también el 15% de interés que le ha pagado la inmobiliaria, que hace un total del 115% de lo que tenía en un principio. En otras palabras, tiene que multiplicar su dinero por 1.15, y esto es válido cualquiera que sea la cantidad inicial. Cuando transcurra otro año habrá ocurrido lo mismo, por lo que tendrá 1.15*1.15; o, dicho de otra forma, 1.15↑2, es decir, 1.3225 veces la cantidad inicial de dinero. En general, al cabo de *y* años, tendrá 1.15↑*y* veces la cantidad inicial.

Si prueba esta orden

```
FOR y=0 TO 100: PRINT y,1000*1.15↑y: NEXT y
```

verá que, incluso empezando con 1000 pesetas, la suma crecerá bastante deprisa y, lo que es más, el ritmo al que aumenta va siendo cada vez mayor. (Pero no se haga ilusiones: para calcular el crecimiento real del valor de su dinero tendría que restar el porcentaje de inflación de ese 15% que le da la inmobiliaria.)

Este tipo de comportamiento, en el que al cabo de cierto intervalo de tiempo una cantidad se multiplica por un número fijo, es lo que se llama *crecimiento exponencial*. El valor final se calcula elevando ese número fijo a la potencia dada por el número de unidades de tiempo.

Supongamos que definimos la siguiente función:

```
10 DEF FN a(x)=a↑x
```

La variable **a** habrá sido definida en una sentencia **LET** (su valor corresponde al tipo de interés, que cambia de vez en cuando).

Hay un valor especial de **a** que hace la función **FN a** particularmente atractiva para el ojo experimentado de un matemático; este valor es el llamado *número e*. El **+2A** posee una función, llamada **EXP**, definida por

EXP *x* es igual a *e*↑*x*

Lamentablemente, el número *e* no es en sí mismo un número demasiado bonito; se trata de un decimal infinito no periódico. Puede ver sus primeros decimales escribiendo la orden:

```
PRINT EXP 1
```

ya que **EXP 1** es igual a *e*↑1, que a su vez es igual a *e*. Por supuesto, esto es sólo una aproximación. No es posible escribir *e* con exactitud.

LN

La función inversa de la exponencial es la *función logarítmica*. El *logaritmo* de base *a* de un número *x* es la potencia a la que hay que elevar *a* para obtener el número *x*; abreviadamente, $\log_a x$. Así, por definición, *a*↑$\log_a x$ es igual a *x*; además, $\log_a$(*a*↑*x*) es igual a *x*. Quizá sepa usted cómo utilizar logaritmos de base 10; son los logaritmos *decimales*. El **+2A** dispone de una función, **LN**, que calcula logaritmos de base *e*, o sea, logaritmos *neperianos* o *naturales*. Para calcular el logaritmo de un número en cualquier otra base se divide el logaritmo neperiano del número por el logaritmo neperiano de la base; es decir, $\log_a x$ es igual a **LN** *x* / **LN** *a*.

PI

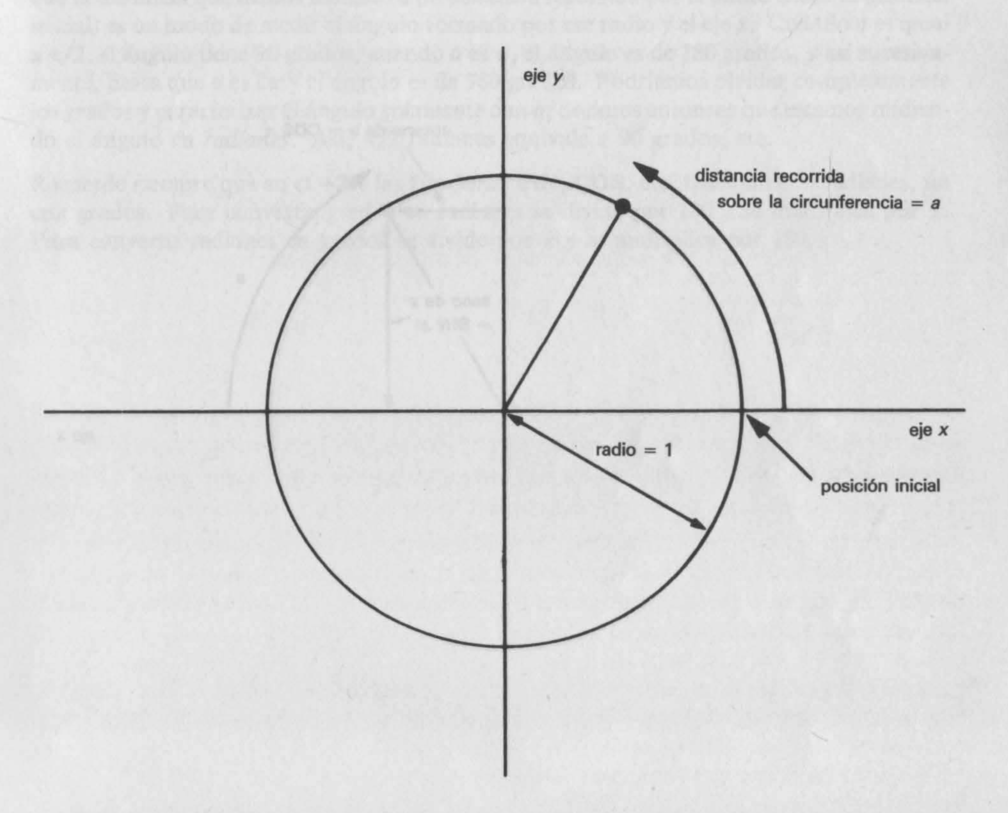
Como es bien sabido, la longitud de la circunferencia se obtiene multiplicando el diámetro por el número π. Este símbolo, π, es una 'p' griega que se lee 'pi'.

Al igual que *e*, π es un decimal infinito no periódico (empieza por 3.1415927). La palabra **PI** representa ese número en el **+2A**. Pruebe lo siguiente:

```
PRINT PI
```

SIN, COS y TAN; ASN, ACS y ATN

Las funciones *trigonométricas* dan una medida de lo que ocurre cuando un punto se mueve alrededor de una circunferencia. Tomemos una circunferencia de radio 1 y un punto que la recorre. El punto empieza en la posición de 'las tres' y se desplaza en el sentido contrario al de las agujas del reloj.

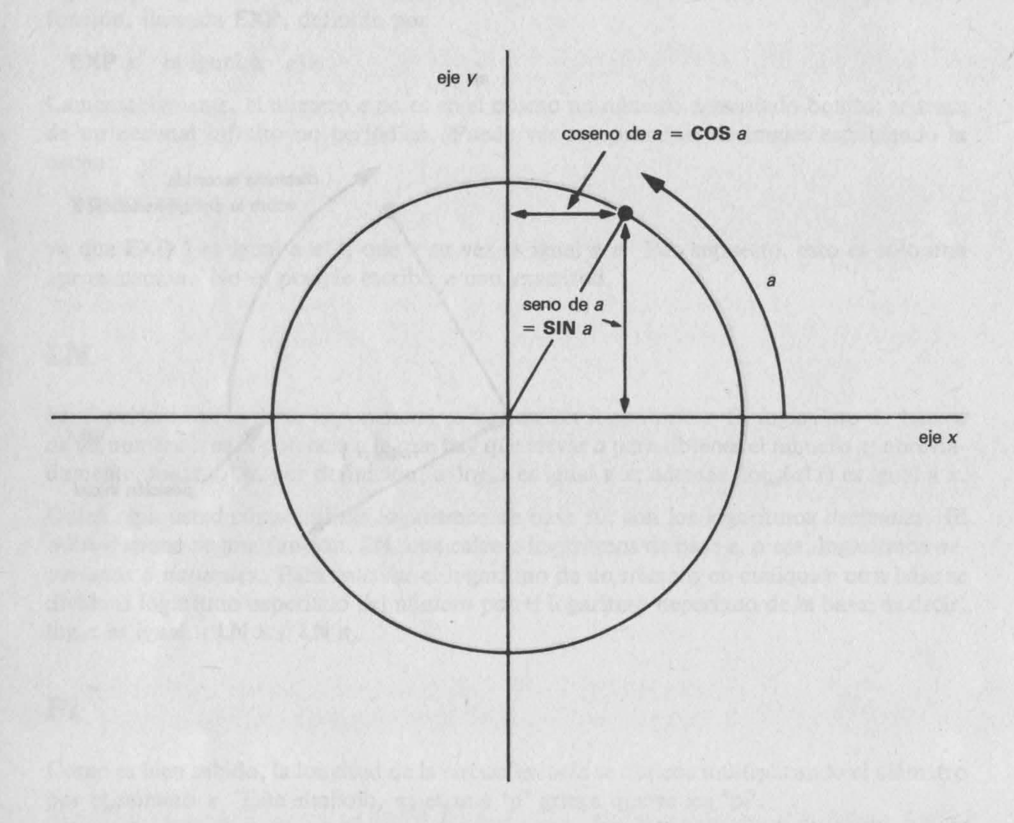


Hemos dibujado dos rectas, llamadas *ejes*, que pasan por el centro de la circunferencia. La que está en la posición de 'las tres' se llama eje x; la que pasa por 'las doce' es el eje y.

Para caracterizar la posición del punto basta con decir qué distancia ha recorrido sobre la circunferencia a partir de su posición inicial; vamos a llamar a a esta distancia. Sabemos que la longitud de la circunferencia es 2π (porque su radio es 1 y su diámetro, por

consiguiente, es 2); así, cuando el punto ha recorrido un cuarto de la longitud de la circunferencia, *a* es igual a $\pi/2$; cuando ha recorrido la mitad, *a* es igual a π; y cuando ha recorrido todo el trecho, *a* es igual a 2π.

Dada la distancia recorrida sobre la circunferencia, *a*, puede interesar conocer la distancia a la que está el punto *a la derecha* del eje *y* y la distancia a la que está por encima del eje *x*. Estas distancias se denominan, respectivamente, *coseno* y *seno* de *a*, y son las calculadas por las funciones **COS** y **SIN** del **+2A**.



Observe que si el punto está a la izquierda del eje *y*, el coseno es negativo; si el punto está por debajo del eje *x*, el seno es negativo.

Otra propiedad interesante es que, una vez que *a* ha crecido hasta 2π, el punto ha llegado a la posición inicial y el seno y el coseno empiezan a tomar otra vez los mismos valores. Es decir, **SIN (*a*+2*PI)** es igual a **SIN *a*** y **COS (*a*+2*PI)** es igual a **COS *a***.

La *tangente* de *a* se define como el seno dividido por el coseno; la función correspondiente en el **+2A** se llama **TAN**.

Algunas veces necesitamos utilizar estas funciones al revés, o sea, averiguar qué valor de *a* ha producido cierto seno, coseno o tangente. Las funciones encargadas de esto se llaman *arco seno* (**ASN** en el **+2A**), *arco coseno* (**ACS**) y *arco tangente* (**ATN**).

Observe el diagrama anterior y fíjese en el radio que une el punto con el centro. Está claro que la distancia que hemos llamado *a* (la distancia recorrida por el punto desde la posición inicial) es un modo de medir el ángulo formado por ese radio y el eje *x*. Cuando *a* es igual a $\pi/2$, el ángulo tiene 90 grados; cuando *a* es π, el ángulo es de 180 grados, y así sucesivamente, hasta que *a* es 2π y el ángulo es de 360 grados. Podríamos olvidar completamente los grados y caracterizar el ángulo solamente con *a*; decimos entonces que estamos midiendo el ángulo en *radianes*. Así, $\pi/2$ radianes equivale a 90 grados, etc.

Recuerde siempre que en el **+2A** las funciones **SIN**, **COS**, etc. trabajan con radianes, no con grados. Para convertir grados en radianes se divide por 180 y se multiplica por π. Para convertir radianes en grados se divide por π y se multiplica por 180.

Sección 11
Números aleatorios

Temas tratados:

RANDOMIZE
RND

En esta sección vamos a estudiar las palabras clave **RND** y **RANDOMIZE**.

En algunos aspectos, **RND** es semejante a una función: realiza cálculos y genera un resultado. En cambio, es peculiar en el sentido de que no necesita argumento.

Cada vez que utilizamos **RND**, el resultado es un nuevo número aleatorio comprendido entre 0 y 1. (Algunas veces puede tomar el valor 0, pero nunca el 1.)

Pruebe lo siguiente:

```
10 PRINT RND
20 GO TO 10
```

¿Puede detectar alguna regla o algún patrón repetitivo en los números? No debería, ya que 'aleatorio' significa 'impredecible', 'sin normas'.

En realidad, **RND** no es perfectamente aleatoria, ya que los valores que genera están tomados de una secuencia fija que consta de 65536 números. Sin embargo, éstos están tan «revueltos» que podemos considerarlos impredecibles a todos los efectos prácticos. Por eso decimos que **RND** es pseudoaleatoria.

RND da un número (pseudo)aleatorio comprendido entre 0 y 1, pero fácilmente se puede transformar ese intervalo en otro cualquiera. Por ejemplo, **5*RND** da valores comprendidos entre 0 y 5; **1.3+0.7*RND** los da en el margen de 1.3 a 2. Cuando necesitemos números enteros podremos utilizar **INT** (sin olvidar que **INT** siempre redondea hacia abajo), como en **1+INT(RND*6)**, expresión que utilizaremos en un programa que simulará el lanzamiento de dados. **RND*6** genera valores que están en el margen de 0 a 6, pero, puesto que realmente nunca llega a 6, **INT(RND*6)** sólo puede ser 0, 1, 2, 3, 4 o 5.

He aquí el programa:

```
10 REM Programa de lanzamiento de dados
20 CLS
30 FOR n=1 TO 2
40 PRINT 1+ INT ( RND *6);" ";
50 NEXT n
60 INPUT a$: GO TO 20
```

Pulse INTRO cada vez que quiera 'lanzar' los dados.

La sentencia **RANDOMIZE** sirve para hacer que **RND** se ponga en marcha a partir de una posición determinada en su secuencia de números, como demuestra este programa:

```
10 RANDOMIZE 1
20 FOR n=1 TO 5: PRINT RND ,: NEXT n
30 PRINT: GO TO 10
```

Tras cada ejecución de **RANDOMIZE 1**, la secuencia de números aleatorios vuelve a arrancar a partir de 0.0022735596. Como parámetro de **RANDOMIZE** se puede usar cualquier número comprendido entre 1 y 65535 para poner en marcha la secuencia **RND** en diferentes posiciones.

Una aplicación de **RANDOMIZE** podría ser la siguiente: supongamos que usted tiene un programa que genera números aleatorios y que no funciona correctamente; puede usar **RANDOMIZE** para hacer que el programa genere siempre los mismos números y así poder estudiar más sistemáticamente su comportamiento.

RANDOMIZE a secas (o **RANDOMIZE 0**) tiene un efecto diferente: elige el punto de partida de forma bastante impredecible. Pruebe el siguiente programa:

```
10 RANDOMIZE
20 PRINT RND: GO TO 10
```

La secuencia que obtenemos aquí no es muy aleatoria, pero esto tiene fácil explicación. **RANDOMIZE** toma como parámetro el tiempo transcurrido desde que se encendió el **+2A**. Como en este programa no transcurre mucho tiempo entre una ejecución de **RANDOMIZE** y la siguiente, **RND** hace más o menos lo mismo todas las veces. Se obtendría una «aleatoriedad» mejor reemplazando **GO TO 10** por **GO TO 20**.

El siguiente programa simula el lanzamiento de monedas y cuenta el número de caras y cruces:

```
10 LET caras=0: LET cruces=0
20 LET moneda= INT ( RND *2)
30 IF moneda=0 THEN LET caras=caras+1
40 IF moneda=1 THEN LET cruces=cruces+1
50 PRINT caras;", ";cruces,
60 IF cruces<>0 THEN PRINT caras/cruces;
70 PRINT: GO TO 20
```

La proporción de caras y cruces debe llegar a ser aproximadamente 1 si se deja que el programa continúe el tiempo suficiente, ya que, a la larga, es de esperar más o menos igual número de caras que de cruces.

Ejercicio

1. Supongamos que usted elige un número comprendido entre 1 y 872 y escribe:

   RANDOMIZE *su-número*

   El siguiente valor generado por **RND** es

   (75∗(*su-número***+1)−1)/65536**

   Compruébelo varias veces.

Sección 12
Matrices

Temas tratados:

Matrices
DIM

Supongamos que tenemos una lista de números (por ejemplo, las notas de diez alumnos de una clase). La forma más obvia de almacenarlos en el **+2A** sería utilizar las variables *m*1, *m*2, *m*3, . . ., *m*10. Sin embargo, el programa necesario para asignar los valores a esas diez variables sería bastante largo y tedioso de escribir:

```
 10 LET m1=75
 20 LET m2=44
 30 LET m3=90
 40 LET m4=38
 50 LET m5=55
 60 LET m6=64
 70 LET m7=70
 80 LET m8=12
 90 LET m9=75
100 LET m10=60
```

Afortunadamente, existe un mecanismo, denominado *matriz*, que permite guardar todos esos valores con un solo nombre de variable. Una matriz es una variable especial que puede contener varios valores, llamados *elementos*; en esto se distingue de las variables ordinarias, que sólo pueden contener uno. Cada elemento de la matriz se identifica por un número, que es el *índice* (o *subíndice*). El índice se escribe entre paréntesis a continuación del nombre de la matriz. En el ejemplo anterior, el nombre global de los elementos de la matriz podría ser *m* (el nombre de las variables matriciales siempre tiene que consistir en una sola letra); por consiguiente, los diez elementos serían *m*(1), *m*(2), *m*(3), . . ., *m*(10).

Los elementos de una matriz reciben también el nombre de *variables indexadas*; las variables que habíamos conocido hasta ahora se llaman *variables sencillas* u *ordinarias*.

Antes de poder utilizar una matriz, es necesario haberle reservado espacio en la memoria del **+2A**; esta operación es lo que se llama *dimensionar* la matriz y se realiza mediante la palabra clave **DIM**. Por ejemplo, la sentencia

```
DIM m(10)
```

reserva espacio en la memoria para una matriz llamada *m* cuyas dimensiones van a ser diez (es decir, equivaldrá a diez variables indexadas). La sentencia **DIM** «inicializa» los elementos de la matriz dándoles el valor 0. Además, si ya existía una matriz con ese mismo nombre, la borra. (Pero no afecta a la variable sencilla *m*, si es que existe; una variable matricial puede coexistir con una variable sencilla numérica del mismo nombre, ya que la matriz es siempre distinguible por su subíndice.)

Los subíndices de los elementos de la matriz pueden ser representados por cualquier expresión numérica que produzca un número válido como subíndice. Gracias a esto, las matrices pueden ser procesadas utilizando bucles **FOR...NEXT**. De ese modo, podemos olvidar el interminable programa que dimos al principio de esta sección y guardar los diez datos mediante el siguiente:

```
10 DIM m(10)
20 FOR n=1 TO 10
30 READ m(n)
40 NEXT n
50 DATA 75,44,90,38,55,64,70,12,75,60
```

(Observe que la sentencia **DIM** tiene que haber sido ejecutada antes de intentar acceder a la matriz.)

Si lo desea, puede usted examinar el contenido de la matriz con el siguiente programa:

```
10 FOR n=1 TO 10
20 PRINT m(n)
30 NEXT n
```

o individualmente con:

```
PRINT m(1)
PRINT m(2)
PRINT m(3)
```
etcétera

También podemos formar matrices con varias dimensiones. En una matriz bidimensional necesitaremos dos números para especificar cada uno de sus elementos, de la misma manera que necesitamos un número de fila y un número de columna para especificar la posición de un carácter en la pantalla. Si imaginamos que los números de línea y de columna (dos dimensiones) describen una página impresa, en la tercera dimensión tendríamos los números de página. Por supuesto, nosotros estamos hablando de matrices numéricas, por lo que los elementos de esa matriz tridimensional no serían caracteres de texto, sino números. Trate de imaginarse los elementos de una matriz tridimensional **v** especificados por **v(*número-de-página*,*número-de-línea*,*número-de-columna*)**.

Por ejemplo, para formar una matriz bidimensional con dimensiones 3 y 6, se utiliza la siguiente sentencia **DIM**:

```
DIM c(3,6)
```

Esa matriz tendrá un total de 3*6=18 variables indexadas:

c(1,1), *c*(1,2), . . . , *c*(1,6)
c(2,1), *c*(2,2), . . . , *c*(2,6)
c(3,1), *c*(3,2), . . . , *c*(3,6)

La misma idea es válida para cualquier número de dimensiones.

Aunque es posible tener una variable sencilla y una matricial con el mismo nombre, no puede haber dos matrices que se llamen igual, aunque tengan diferente número de dimensiones.

Hasta ahora hemos descrito las *matrices numéricas*. También existen *matrices literales*, cuyos elementos son, naturalmente, cadenas literales. Las cadenas de una matriz se distinguen de las cadenas sencillas en que son de longitud fija y en que la asignación de valores se realiza por el procedimiento de Procrustes (¿lo recuerda?): si el valor es demasiado largo, se lo recorta; si es demasiado corto, se lo complementa con espacios por la derecha.

El nombre de toda matriz literal consiste en una sola letra seguida de **$**. A diferencia de lo que ocurría con las matrices numéricas, una matriz literal y una variable literal sencilla no pueden tener el mismo nombre.

Supongamos, pues, que queremos una matriz *a*$ que pueda albergar cinco cadenas. Debemos decidir qué longitud van a tener las cadenas. Por ejemplo, una matriz para cinco cadenas de diez caracteres cada una se dimensiona con:

DIM a$(5,10) (escriba esta orden)

Esta sentencia prepara una matriz de 5*10 *caracteres*, que podemos manejar individualmente o por filas completas:

a$(1)=*a*$(1,1)*a*$(1,2). . .*a*$(1,10)
a$(2)=*a*$(2,1)*a*$(2,2). . .*a*$(2,10)
a$(3)=*a*$(3,1)*a*$(3,2). . .*a*$(3,10)
a$(4)=*a*$(4,1)*a*$(4,2). . .*a*$(4,10)
a$(5)=*a*$(5,1)*a*$(5,2). . .*a*$(5,10)

Si al usar la matriz especificamos el mismo número de subíndices (dos en este caso) que dimensiones hay en la sentencia **DIM**, obtenemos un solo carácter. Pero si omitimos el último, la matriz da la fila entera (una cadena de longitud fija). Así que, por ejemplo, *a*$(2,7) es el séptimo carácter de la cadena *a*$(2). Utilizando la notación de disección de cadenas (Sección 8 de este capítulo) también podríamos escribir *a*$(2)(7) en lugar de *a*$(2,7). Ahora escriba:

LET a$(2)="1234567890"

y

PRINT a$(2), a$(2,7)

El resultado será:

1234567890 7

El último subíndice (el que se puede omitir), también puede tener la estructura de los parámetros de la disección de cadenas. Por ejemplo,

a$(2,4 TO 8) es igual a **a$(2)(4 TO 8)** que a su vez es igual a **"45678"**

Recuerde: en una matriz literal, todas las cadenas tienen la misma longitud.

La sentencia **DIM** tiene un parámetro adicional (el último) que especifica esa longitud. Al escribir una variable indexada perteneciente a una matriz literal, podemos incluir un número adicional (o bien un *disector de cadenas*), que corresponderá al último parámetro de la sentencia **DIM**.

Si al dimensionar una matriz literal sólo especificamos un parámetro, la matriz se comporta como variable sencilla de longitud fija. Escriba:

DIM a$(10)

y podrá comprobar que **a$** funciona igual que una variable literal ordinaria, con la única diferencia de que su longitud siempre será 10 y los valores le serán asignados por el método de Procrustes.

Ejercicio

1. Utilice las sentencias **READ** y **DATA** para formar una matriz **m$** en la cual **m$(n)** sea el nombre del **n**-ésimo mes. (*Sugerencia.* Suponiendo que usted tenga la costumbre de escribir 'setiembre' en vez de 'septiembre', la sentencia **DIM** necesaria es **DIM m$(12,9)**.) Compruebe todos los valores de **m$(n)** haciendo que los escriba un bucle **FOR...NEXT**.

Sección 13
Condiciones

Temas tratados:

AND, OR
NOT

Vimos en la Sección 3 de este capítulo que la sentencia **IF** toma la forma:

IF *condición* **THEN . . .**

Las condiciones eran entonces relaciones (construidas a base de =, <, >, <=, >= y <>) que comparaban dos números o dos cadenas. También podemos combinar varias relaciones utilizando los operadores lógicos: **AND** ('y'), **OR** ('o') y **NOT** ('no').

Una relación combinada mediante **AND** con otra relación da el valor 'verdadero' siempre que **ambas** sean verdaderas; por ejemplo, podríamos tener una línea del siguiente estilo:

IF a$="s" AND x>0 THEN PRINT "resultado"

con la que BASIC sólo escribe el texto si **a$** es igual a **"s"** y **x** es mayor que cero.

Una relación combinada mediante **OR** con otra relación da el valor 'verdadero' siempre que **al menos una** de ellas sea verdadera. (El resultado también es verdadero si las dos relaciones son verdaderas.)

La relación **NOT** invierte el valor lógico: **NOT** *relación* es 'verdadero' siempre que la *relación* es falsa, y 'falso' cuando la *relación* es verdadera.

Las *expresiones lógicas* consisten en combinaciones de **AND**, **OR** y **NOT**, de la misma forma que las expresiones numéricas son combinaciones de +, −, *, /, etc. También se puede colocarlas entre paréntesis si es necesario. Las operaciones lógicas tienen un orden de prioridad, lo mismo que +, −, *, / y ↑. **OR** es la menos prioritaria; en orden de prioridad ascendente le siguen **AND** y **NOT**.

NOT es en realidad una función, con un argumento y un resultado, pero su prioridad es muy inferior a la de las otras funciones. Por lo tanto, su argumento no necesita paréntesis, a menos que sea en sí una operación lógica (construida con **AND**, **OR** o ambas). **NOT a=b** significa lo mismo que **NOT (a=b)**, y, por supuesto, lo mismo que **a<>b**.

<> es la negación de =, en el sentido de que <> es verdadero si y sólo si = es falso.

En otras palabras

a<>b es lo mismo que **NOT a=b**

y además

NOT a<>b es lo mismo que **a=b**

Si lo piensa, se dará cuenta de que >= y <= son la negación de < y >, respectivamente. Por consiguiente, para invertir el valor de una relación se puede escribir **NOT** delante de ella o sustituirla por su negación.

Por otra parte,

NOT (*primera-expresión-lógica* AND *segunda-expresión-lógica*)

es lo mismo que

NOT (*primera-expresión-lógica*) OR NOT (*segunda-expresión-lógica*)

Además,

NOT (*primera-expresión-lógica* OR *segunda-expresión-lógica*)

es lo mismo que

NOT (*primera-expresión-lógica*) AND NOT (*segunda-expresión-lógica*)

Utilizando estas reglas, la negación de una expresión se puede reducir a una expresión en la que **NOT** esté aplicada directamente a relaciones. En último extremo, **NOT** no es imprescindible, aunque en la práctica facilita la programación al hacer más inteligibles las expresiones lógicas.

Lo que resta de esta sección es bastante complicado; si no le interesa mucho este tema, puede omitir su lectura y pasar directamente a la Sección 14.

Pruebe la siguiente orden:

PRINT 1=2, 1<>2

que aparentemente debería producir un error de sintaxis. En realidad, el ordenador no sabe qué es eso de 'valor lógico'; en lugar de los valores 'verdadero' y 'falso' utiliza números ordinarios, que están sujetos a unas cuantas reglas:

(i) Cuando el ordenador evalúa las relaciones construidas a base de =, <, >, <=, >= y <>, obtiene un valor numérico que es 1 cuando la relación es verdadera y 0 cuando la relación es falsa. Por eso la anterior orden **PRINT** escribió 0 para **1=2**, que es una relación falsa, y 1 para **1<>2**, que es verdadera.

(ii) En la sentencia

IF *condición* **THEN ...**

la *condición* puede ser en realidad una expresión numérica cualquiera. Si su valor es 0, el ordenador actúa como si se tratase de una expresión lógica con valor 'falso'; si el valor de *condición* es distinto de 0, el ordenador la considera 'verdadera'. Por tanto, la anterior sentencia **IF** es exactamente equivalente a

IF *condición*<>**0 THEN ...**

(iii) **AND, OR** y **NOT** son también operaciones aplicables a expresiones numéricas cualesquiera:

x AND y vale { **x** si **y** es verdadero (distinto de cero)
0 (falso) si **y** es falso (cero)

x OR y vale { 1 (verdadero) si **y** es verdadero (distinto de cero)
x si **y** es falso (cero)

NOT x vale { 0 (falso) si **x** es verdadero (distinto de cero)
1 (verdadero) si **x** es falso (cero)

(Observe que 'verdadero' significa 'distinto de cero' cuando estamos comprobando un valor dado, pero que significa '1' cuando estamos produciendo un valor nuevo.)

Ahora pruebe este programa:

```
10 INPUT a
20 INPUT b
30 PRINT (a AND a>= b)+(b AND a< b)
40 GO TO 10
```

En cada vuelta, este programa escribe el mayor de los dos números introducidos.

Si se para a pensarlo, se dará cuenta de que

x AND y

se podría leer así:

x si **y** (de lo contrario, el resultado es 0)

y de que

x OR y

significa lo mismo que

x a no ser que **y** (en cuyo caso el resultado es 1)

Una expresión en la que se utilice **AND** u **OR** de esta forma es lo que se llama *expresión condicional*. Un ejemplo con **OR** podría ser:

```
LET total=precio sin impuesto*(1.12 OR v$="Exento de IVA")
```

Observe que **AND** tiende a asociarse con la suma (porque su valor por defecto es 0) y **OR** con la multiplicación (porque su valor por defecto es 1).

También podemos formar expresiones condicionales cuyo valor sea una cadena literal, pero sólo utilizando **AND**:

x$ AND y vale { **x$** si **y** es distinto de cero; **""** (la cadena vacía) si **y** es cero }

de modo que **x$ AND y** significa '**x$** si **y** (de lo contrario, su valor es la cadena vacía)'.

Pruebe este programa, que capta dos cadenas y las pone en orden alfabético:

```
10 INPUT "Escriba dos cadenas" 'a$,b$
20 IF a$> b$ THEN LET c$=a$: LET a$=b$: LET b$=c$
30 PRINT a$;" ";("<" AND a$< b$)+("=" AND a$=b$);" ";b$
40 GO TO 10
```

Sección 14
El juego de caracteres

Temas tratados:

CODE, CHR$
POKE, PEEK
USR
BIN

Las letras, dígitos, espacios, signos de puntuación, etc. que pueden formar parte de las cadenas literales se llaman *caracteres* y componen el *juego de caracteres* del **+2A**. En su mayor parte, estos caracteres son símbolos simples, pero algunos representan palabras completas, tales como **PRINT**, **STOP**, **<>**, etc.

Hay un total de 256 caracteres, cada uno de ellos identificado por un código comprendido entre 0 y 255 (en la Sección 28 de este capítulo damos la lista completa). Para efectuar la conversión entre códigos y caracteres disponemos de dos funciones: **CODE** y **CHR$**.

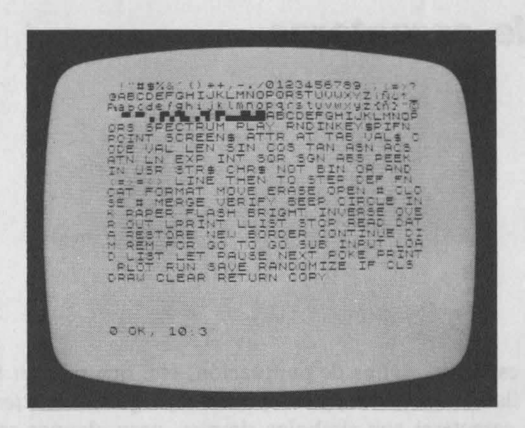
CODE se aplica a una cadena y da el código de su primer carácter (o, si se trata de la cadena vacía, da el número 0).

CHR$ se aplica a un número y da la cadena de un único carácter cuyo código es ese número.

El siguiente programa escribe el juego de caracteres del **+2A** (del 32.º en adelante):

```
10 FOR a=32 TO 255: PRINT CHR$ a;: NEXT a
```

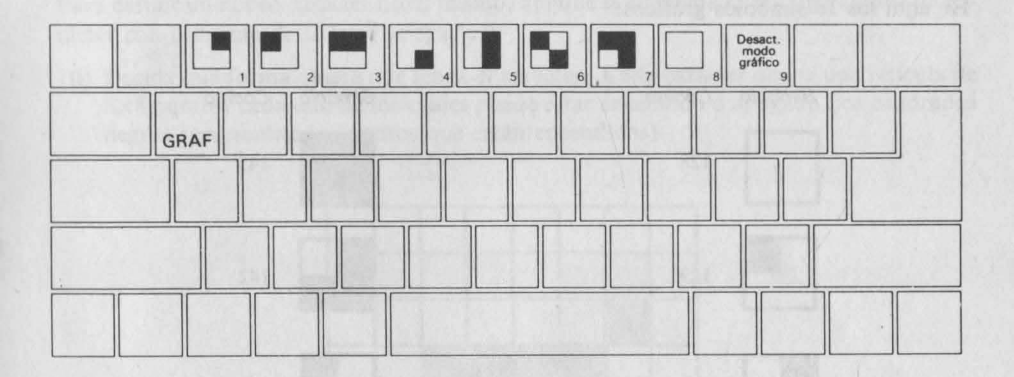
En la pantalla aparecerá lo siguiente:



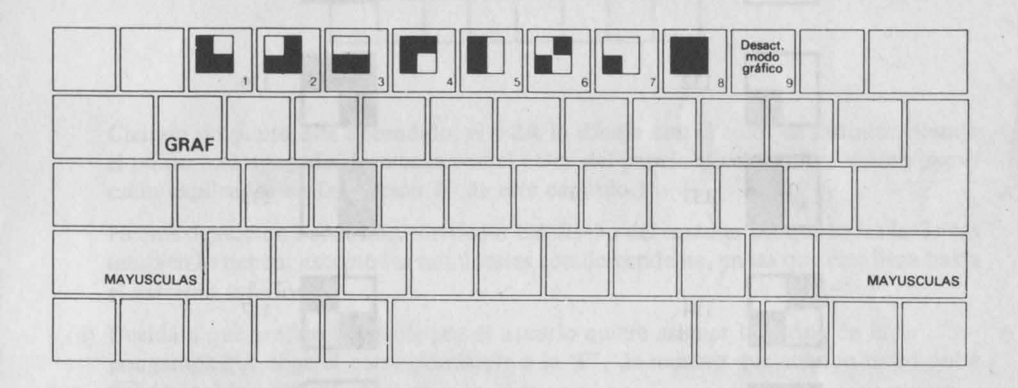
El juego de caracteres

Como puede ver, el juego de caracteres consta del espacio, 15 símbolos y signos de puntuación, los diez dígitos, otros siete símbolos, las mayúsculas, otros seis símbolos (entre los que se encuentra la 'Ñ'), las minúsculas y cinco símbolos más (entre los que se encuentra la 'ñ'). Todos ellos (excepto 'Pt', '©', '¡', '¿', 'ñ' y 'Ñ') están tomados de un juego de caracteres estándar, muy conocido y ampliamente utilizado: el juego ASCII (*American Standard Code for Information Interchange*, 'Código norteamericano estándar para el intercambio de la información'). Los caracteres del juego ASCII están identificados por códigos, y esos códigos son los que emplea el **+2A**.

Los demás caracteres no forman parte del juego ASCII, sino que son específicos de los ordenadores ZX Spectrum. Los primeros de ellos son un espacio y 15 combinaciones de cuadrados blancos y negros; éstos son los llamados *símbolos gráficos* y pueden ser usados para dibujar. El usuario puede introducirlos por el teclado seleccionando el denominado *modo gráfico* (modo **G**). Al pulsar la tecla [GRAF] se activa el modo gráfico; para obtener los símbolos gráficos se pulsa a continuación las teclas numéricas (de la [1] a la [8]).



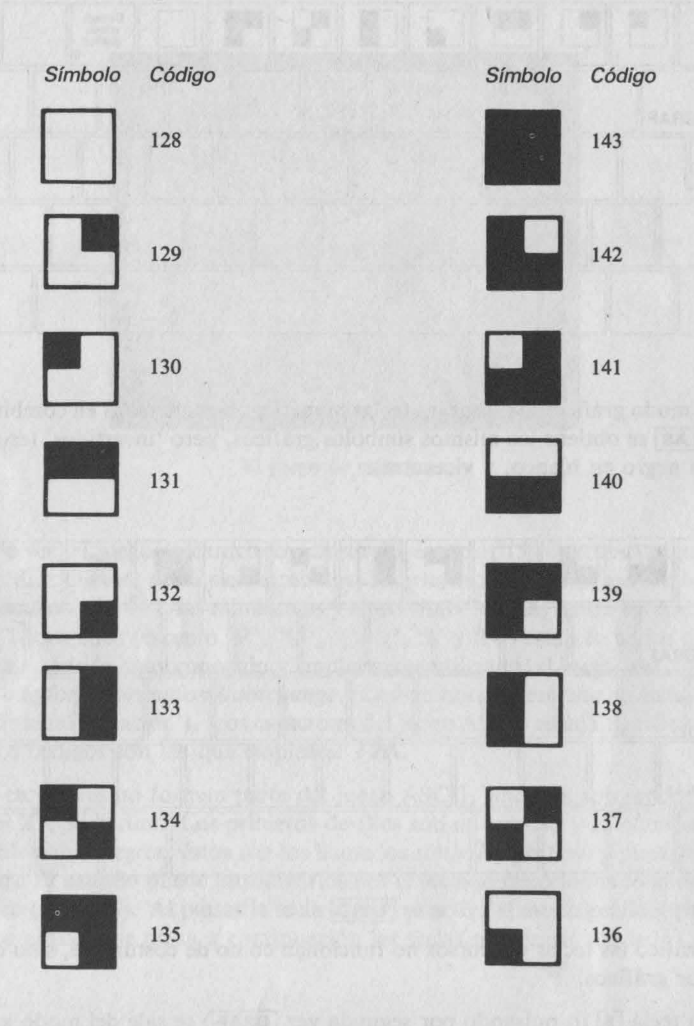
Estando en modo gráfico, al pulsar las teclas numéricas mencionadas en combinación con [MAYUSCULAS] se obtiene los mismos símbolos gráficos, pero 'invertidos' (es decir, convirtiendo lo negro en blanco, y viceversa).



En modo gráfico las teclas del cursor no funcionan como de costumbre, sino que producen símbolos gráficos.

Pulsando la tecla [9] (o pulsando por segunda vez [GRAF]) se sale del modo gráfico y se vuelve al normal. Pulsando la tecla [0] se borra el carácter que está a la izquierda del cursor.

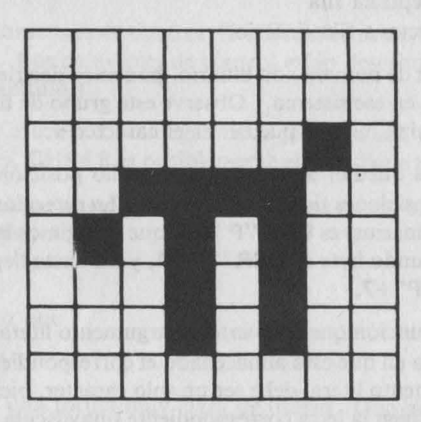
He aquí los 16 símbolos gráficos:



Los siguientes caracteres del juego del **+2A** son, aparentemente, una repetición de las mayúsculas de la 'A' a la 'S'. En realidad, se trata de caracteres cuya forma podemos rediseñar (sin embargo, mientras no los rediseñemos, tendrán por defecto la forma de esas letras mayúsculas); se les llama *gráficos definibles por el usuario* (o, abreviadamente, *gráficos de usuario*). Para introducirlos por el teclado primero se activa el modo gráfico y luego se pulsa una tecla alfabética (de la [A] a la [S]).

Para definir un nuevo carácter usted mismo, aplique la siguiente receta (que define un carácter con la forma de la letra griega 'π'):

(i) Decida qué forma quiere que tenga el carácter. Cada carácter ocupa una retícula de 8×8 puntos, cada uno de los cuales puede estar encendido o apagado (los cuadrados negros representan los puntos que están encendidos).



Cuando un punto está encendido, el **+2A** lo dibuja con el color de la tinta; cuando el punto está apagado, lo dibuja con el color del papel. (Los términos *tinta* y *papel* están explicados en la Sección 16 de este capítulo.)

Hemos dejado un borde libre alrededor del diseño del carácter porque todas las letras también lo tienen, excepto las minúsculas con descendente, en las que éste llega hasta el extremo inferior.

(ii) Decida a qué gráfico definible por el usuario quiere asignar la forma de la 'π'. Supongamos que elige el correspondiente a la 'P', de manera que cuando usted pulse [P] (en modo gráfico) obtenga π.

(iii) Almacene el nuevo diseño. Cada gráfico definido por el usuario se almacena en forma de una sucesión de ocho números, uno para cada fila. Podemos especificar estos números en un programa escribiendo la palabra **BIN** seguida de 8 ceros o unos (0 para el papel, 1 para la tinta). De esta forma, los ocho números que describen nuestro carácter π son:

```
BIN 00000000  primera fila (superior)
BIN 00000000  segunda fila
BIN 00000010  tercera fila
BIN 00111100  cuarta fila
BIN 01010100  quinta fila
BIN 00010100  sexta fila
BIN 00010100  séptima fila
BIN 00000000  octava fila (inferior)
```

(Si conoce el sistema de numeración binario, podemos decirle que **BIN** se utiliza para escribir los números en ese sistema.) Observe este grupo de números binarios con los ojos entornados: quizá incluso pueda ver el carácter π.

Estos ocho números quedan almacenados en ocho posiciones (bytes) de memoria. Cada una de estas posiciones tiene una *dirección*. La dirección del primer byte (o grupo de ocho dígitos binarios) es **USR "P"** [porque escogimos la 'P' en el apartado (ii)]. La dirección del segundo byte es **USR "P"+1**, y así hasta llegar al octavo byte, cuya dirección es **USR "P"+7**.

Aquí, **USR** es una función que convierte un argumento literal en la dirección del primer byte de memoria en que está almacenado el correspondiente gráfico definido por el usuario. El argumento literal debe ser un solo carácter, bien el propio gráfico definido por el usuario, bien la letra correspondiente (mayúscula o minúscula). **USR** tiene otro significado cuando su argumento es un número, pero de eso hablaremos más adelante.

Pues bien, aunque no haya conseguido seguirnos en el proceso que acabamos de explicar, pruebe el siguiente programa, que realiza la definición del carácter:

```
10 FOR n=0 to 7
20 READ fila: POKE USR "P"+n, fila
30 NEXT n
40 DATA BIN 00000000
50 DATA BIN 00000000
60 DATA BIN 00000010
70 DATA BIN 00111100
80 DATA BIN 01010100
90 DATA BIN 00010100
100 DATA BIN 00010100
110 DATA BIN 00000000
```

La sentencia **POKE** almacena directamente un número en una posición de memoria, eludiendo el mecanismo normalmente usado por BASIC. Lo inverso de **POKE** es **PEEK**, función que permite leer (sin modificarlo) el contenido de las posiciones de memoria. En la Sección 24 de este capítulo describiremos **PEEK** y **POKE** más detalladamente.

Continuando con el juego de caracteres, los siguientes son los *códigos de palabras clave* o, abreviadamente, *claves.*

Habrá notado que en el primer programa de esta sección no hemos escrito los 32 primeros caracteres (códigos 0 al 31); son los *caracteres de control.* No producen ningún carácter visible, sino que nos permiten controlar la imagen que se forma en la pantalla y algunas otras funciones del **+2A**.

(Si intentamos escribir caracteres de control, el **+2A** muestra un ? en la pantalla para indicar que no los entiende. Los caracteres de control están descritos más detalladamente en la Sección 28 de este capítulo.)

Tres de los códigos que afectan a la imagen de la pantalla son el 6, el 8 y el 13. Vamos a explicarlos. De los tres, **CHR$ 8** es posiblemente el único que tendrá interés para usted.

CHR$ 6 inserta espacios exactamente de la misma forma que lo hace la coma en las sentencias **PRINT**. Por ejemplo,

```
PRINT 1; CHR$ 6;2
```

produce el mismo efecto que

```
PRINT 1,2
```

Obviamente, ésta no es una forma muy clara de usarlo. Una manera más comprensible sería:

```
LET a$="1"+CHR$ 6+"2"
PRINT a$
```

CHR$ 8 es el *espacio hacia atrás* o *retroceso del cursor*: desplaza la posición de escritura un lugar hacia la izquierda. Pruebe la siguiente orden:

```
PRINT "1234"; CHR$ 8;"5"
```

cuyo efecto final es escribir

```
1235
```

CHR$ 13 es *línea nueva*: desplaza la posición de escritura al comienzo de la línea siguiente. La pantalla reconoce también los códigos 16 al 23, que están explicados en las Secciones 15 y 16 de este capítulo. (La lista completa de todos los códigos se encuentra en la Sección 28.)

Teniendo en cuenta todos los caracteres «visibles», podemos ampliar el concepto de 'orden alfanumérico' a cadenas que contengan caracteres cualesquiera, no sólo letras. Para ello debemos considerar un alfabeto ampliado que, en lugar de constar de 26 letras, sea la lista de todos los 256 caracteres, en el mismo orden que sus códigos. Por ejemplo, las siguientes cadenas están, para el **+2A**, en orden alfabético. (Observe que, curiosamente, las letras minúsculas van detrás de todas las mayúsculas, de forma que la 'a' es posterior a la 'Z'; además, los espacios son significativos.)

```
CHR$ 3+"ZOOLOGICO"
CHR$ 8+"AARRR"
" AAAARRR!"
"(Nota entre paréntesis)"
"100"
"12995 inc. IVA"
"AAARRR"
"AaaRRR"
"PRINT"
"Sigmund Freud"
"Zoo"
"[interpolación]"
"aaarrr"
"aaasss"
"derecho"
"zoo"
"zoología"
```

La regla para ordenar dos cadenas es la siguiente. Primero se compara el primer carácter de una con el primer carácter de la otra. Si son diferentes, uno de los dos caracteres tendrá un código menor que el otro; la primera cadena en orden «alfabético» es la que empieza por el carácter cuyo código es menor. En cambio, si los dos caracteres son iguales, se pasa al segundo carácter de las dos cadenas y se realiza con ellos la comparación; y así sucesivamente hasta que los caracteres comparados sean diferentes. Si, en este proceso, una de las cadenas termina antes que la otra sin que se haya detectado diferencia entre los caracteres, la cadena más corta es la primera; de lo contrario, las dos cadenas son iguales.

Las relaciones =, <, >, <=, >= y <> son aplicables tanto a cadenas como a números: < significa 'es anterior a' y > 'es posterior a', de modo que, por ejemplo, las relaciones

```
"AA RRR"<"AAARRR"
"AAARRR">"AA RRR"
```

son ambas verdaderas.

<= y >= funcionan de la misma forma que con números, y por lo tanto el valor de la relación

```
"Esta cadena"<="Esta cadena"
```

es 'verdadero', mientras que el de

```
"Esta cadena"<"Esta cadena"
```

es 'falso'.

Para experimentar con todo esto, ejecute el programa siguiente, que capta dos cadenas y las ordena:

```
10 INPUT "Escriba dos cadenas:",a$,b$
20 IF a$>b$ THEN LET c$=a$: LET a$=b$: LET b$=c$
30 PRINT a$;" ";
40 IF a$< b$ THEN PRINT "<";: GO TO 60
50 PRINT "=";
60 PRINT " ";b$
70 GO TO 10
```

Observe que, tanto en este programa como en el del final de la Sección 13, hemos tenido que usar **c$** (línea 20) para poder intercambiar los valores de **a$** y **b$**. ¿Comprende por qué con

```
LET a$=b$: LET b$=a$
```

no habríamos obtenido el efecto deseado?

El siguiente programa define gráficos de forma que las siguientes teclas produzcan las piezas del ajedrez:

[B] para el alfil
[K] para el rey
[R] para la torre
[Q] para la reina
[P] para el peón
[N] para el caballo

Programa:

```
  5 LET b=BIN 01111100: LET c=BIN 00111000: LET d=BIN 00010000
 10 FOR n=1 TO 6: READ p$: REM 6 piezas
 20 FOR f=0 TO 7: REM leer 8 bytes
 30 READ a: POKE USR p$+f,a
 40 NEXT f
 50 NEXT n
100 REM alfil
110 DATA "b", 0, d, BIN 00101000, BIN 01000100
120 DATA BIN 01101100, c, d, 0
130 REM rey
140 DATA "k", 0, d, c, d
150 DATA c, BIN 01000100, c, 0
160 REM torre
170 DATA "r", 0, BIN 01010100, b, c
180 DATA c, b, b, 0
```

```
190 REM reina
200 DATA "q", 0, BIN 01010100, BIN 00101000, d
210 DATA BIN 01101100, b, b, 0
220 REM peón
230 DATA "p", 0, 0, d, c
240 DATA c, d, b, 0
250 REM caballo
260 DATA "n", 0, d, c, BIN 01111000
270 DATA BIN 00011000, c, b, 0
```

Observe que en las sentencias **DATA** anteriores hemos puesto **0** en lugar de **BIN 00000000**. Cuando haya ejecutado el programa, puede ver las piezas pulsando [GRAF] y cualquiera de las teclas: [B], [K], [R], [Q], [P] o [N].

Ejercicios

1. Imagine el espacio para un símbolo dividido en cuatro cuartos. Si cada cuarto puede ser blanco o negro, hay $2^4=16$ posibilidades. Búsquelas todas en el juego de caracteres.

2. Ejecute este programa:

```
10 INPUT a
20 PRINT CHR$ a;
30 GO TO 10
```

   Puede comprobar que **CHR$** redondea el número **a** al entero más cercano. Si **a** no está en el margen de 0 a 255, el programa se detiene y BASIC emite el mensaje de error '**B ENTERO EXCEDE MARGEN**'.

3. ¿Cuál de las dos cadenas siguientes es menor?

```
"GATO"
"gato"
```

Sección 15
Más sobre PRINT e INPUT

Temas tratados:

CLS
Elementos de **PRINT**
Expresiones (numéricas y literales)
TAB
AT
, ; ' (separadores de **PRINT**)
Elementos de **INPUT**
LINE
Rodadura de la pantalla
SCREEN$

Hemos utilizado la orden **PRINT** bastantes veces, así que usted ya tendrá una idea bastante buena de cómo funciona. Las expresiones cuyos valores escribimos con **PRINT** son los *elementos de* **PRINT**. Pueden estar separados entre sí por comas, apóstrofos o signos de punto y coma. Todos estos signos de puntuación reciben el nombre de *separadores de* **PRINT**. Un elemento de **PRINT** puede ser también 'nada', y esto explica qué ocurre cuando en una instrucción **PRINT** ponemos solamente la palabra clave.

Hay otros dos tipos de elementos de **PRINT** que sirven para comunicar al **+2A** *dónde* debe escribir, no *qué* debe escribir. Por ejemplo, la instrucción

```
10 PRINT AT 11,16;"*"
```

escribe un asterisco en el centro de la pantalla. Esto se debe a que

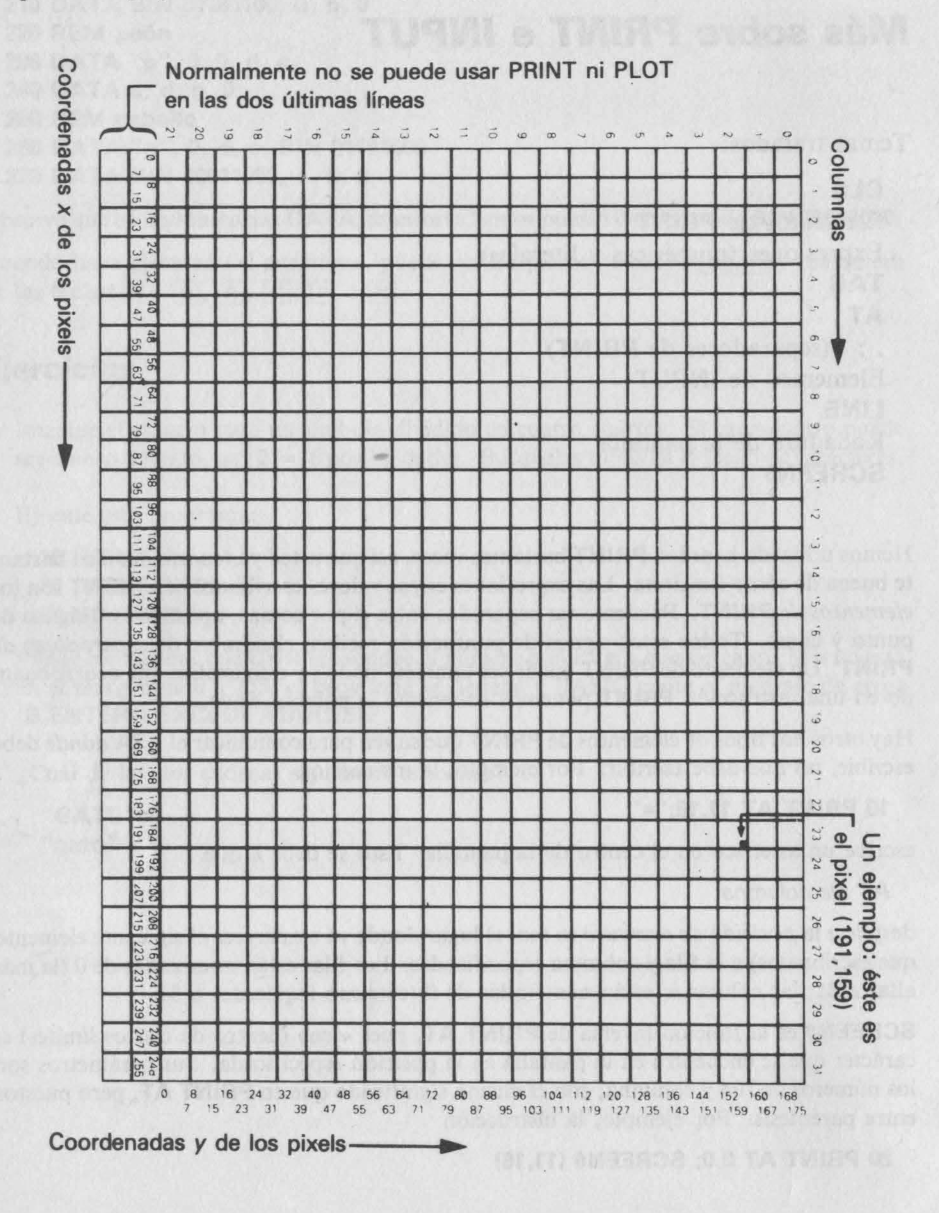
AT *fila,columna*

desplaza la *posición de escritura* (o sea, el lugar donde va a aparecer el siguiente elemento que escribamos) a la fila y columna especificadas. Las filas están numeradas de 0 (la más alta) a 21; las columnas están numeradas de 0 (extremo izquierdo) a 31.

SCREEN$ es la función inversa de **PRINT AT**, pues «lee» (dentro de ciertos límites) el carácter que se encuentra en la pantalla en la posición especificada. Sus parámetros son los números de fila y columna, con el mismo significado que en **PRINT AT**, pero puestos entre paréntesis. Por ejemplo, la instrucción

```
20 PRINT AT 0,0; SCREEN$ (11,16)
```

lee el asterisco que escribimos antes en el centro de la pantalla y luego lo reproduce en la posición 0,0 (el extremo superior izquierdo).



Los códigos de palabras clave son leídos normalmente (como caracteres sencillos); los espacios son leídos como tales. En cambio, si se intenta leer caracteres definidos por el usuario, caracteres gráficos o líneas dibujadas por **PLOT**, **DRAW** o **CIRCLE**, la función **SCREEN$** da la cadena vacía. Lo mismo ocurre si se ha usado **OVER** para crear un carácter compuesto. (Las palabras clave **PLOT**, **DRAW**, **CIRCLE** y **OVER** están descritas en las Secciones 16 y 17 de este capítulo.)

La función

TAB *columna*

escribe los espacios necesarios para desplazar la posición de escritura hasta la columna especificada. Intenta no cambiar de línea pero, si ello la obliga a retroceder, prefiere saltar a la línea siguiente. Observe que el **+2A** toma el número de columna 'módulo 32' (es decir, divide por 32 y toma el resto), de forma que **TAB 33** significa lo mismo que **TAB 1**.

Por ejemplo,

```
PRINT TAB 30; 1; TAB 12; "Contenido"; AT 3,1; "Capitulo"; TAB 25; "Pagina"
```

sería una forma de escribir la cabecera de la primera página de un libro.

Pruebe ahora este programa

```
10 FOR n=0 TO 20
20 PRINT TAB 8*n;n;
30 NEXT n
```

Esto muestra lo que entendemos por 'tomar el número de columna módulo 32'.

Pruebe el programa después de cambiar el **8** de la línea 20 por un **6**.

Observe los siguientes detalles:

(i) Las cláusulas **TAB** y los elementos de **PRINT** normalmente deben terminar en punto y coma. No es que no se pueda poner comas (o nada, al final de la sentencia), pero, después de haber seleccionado tan cuidadosamente la posición de escritura, no nos interesa volver a desplazarla inmediatamente.

(ii) No se puede escribir en las dos últimas líneas de la pantalla (22 y 23) porque están reservadas para órdenes, captación de datos con **INPUT**, mensajes de error, informes, etc. Así pues, cuando en lo sucesivo hablemos de 'última línea' normalmente nos estaremos refiriendo a la línea 21.

(iii) Se puede utilizar **AT** para situar la posición de escritura en una posición en la que ya haya algo escrito (el nuevo elemento sencillamente reemplazará al antiguo).

Otra sentencia relacionada con **PRINT** es **CLS**, cuyo efecto es borrar toda la pantalla.

Cuando al escribir llegamos a la parte inferior de la pantalla, el texto empieza a desplazarse hacia arriba como si se tratase de la hoja de papel en una máquina de escribir. Para

comprobar cómo funciona este mecanismo de *rodadura*, seleccione la opción **Pantalla** en el menú de edición (descrito en el capítulo 6) y luego escriba:

```
CLS: FOR n=1 TO 30: PRINT n: NEXT n
```

Cuando se llena la pantalla, el **+2A** se detiene y emite el mensaje '**¿MAS?**' en la parte inferior de la misma. Ahora puede usted observar a su gusto los 22 primeros números. Cuando haya terminado, pulse [S] (para 'sí') y el **+2A** mostrará la siguiente tanda de números. En realidad, da lo mismo pulsar [S] que cualquier otra tecla, a excepción de [N], la barra espaciadora y [BREAK]. Estas últimas hacen que el programa se detenga y que el ordenador emita el informe '**D BREAK – CONT repite**'.

La sentencia **INPUT** es capaz de hacer mucho más de lo que le hemos exigido hasta ahora. Ya hemos visto sentencias **INPUT** similares a

```
INPUT "¿Cuantos años tienes?", edad
```

en la cual el **+2A** escribe la frase **¿Cuantos años tienes?** en la pantalla inferior y queda a la espera de que el usuario introduzca un número. Sin embargo, una sentencia **INPUT** puede incluir elementos y separadores, exactamente igual que las sentencias **PRINT**. **¿Cuantos años tienes?** y **edad** son elementos de **INPUT**.

Los elementos de **INPUT** son generalmente iguales a los de **PRINT**, aunque hay algunas diferencias importantes:

Primero, un elemento adicional obvio de **INPUT** es la variable a la que se asigna valor en la sentencia (en el ejemplo anterior, **edad**). La regla es que si un elemento de **INPUT** comienza con una letra, BASIC considera que se trata de una variable cuyo valor se va a introducir por el teclado.

Podría parecer que esto implica que no podremos escribir los valores de las variables como parte de un mensaje inductor. (*Mensaje inductor* es el que se escribe en una sentencia **INPUT** para indicar al usuario cuál es la naturaleza de los datos que debe introducir.) Sin embargo, esto se resuelve poniendo la variable entre paréntesis. Cualquier expresión que empiece con una letra debe ir entre paréntesis si queremos que **INPUT** la escriba como parte del mensaje inductor.

Cualquier clase de elemento de **PRINT** al que no afecten estas reglas es también un elemento de **INPUT**. Veamos un ejemplo:

```
LET mi edad = INT (RND*100): INPUT ("Tengo ";mi edad;" años.");
"¿Cuantos años tienes tu?", tu edad
```

Puesto que **mi edad** está entre paréntesis, **INPUT** escribe su valor. En cambio, **tu edad** no va entre paréntesis, y por eso **INPUT** entiende que se trata de la variable a la que debe asignar el valor captado.

Todo lo que escribe una sentencia **INPUT** va a la pantalla inferior, que actúa de forma bastante independiente de la pantalla superior. En particular, sus líneas se numeran a partir de la primera línea de la propia pantalla inferior, aun cuando ésta haya crecido hacia arriba en el monitor (lo cual ocurre cuando hay que escribir muchos datos en respuesta a **INPUT**). Cualquier cosa que haga la pantalla inferior durante la ejecución de **INPUT**, su tamaño siempre revertirá a las dos líneas habituales en cuanto se detenga el programa y volvamos al modo de edición.

Para ver cómo funciona **AT** en una sentencia **INPUT**, pruebe lo siguiente:

```
10 INPUT "Esta es la linea 1",a$; AT 0,0;"Esta es la linea 0",a$;
   AT 2,0;"Esta es la linea 2",a$; AT 1,0;"Esta sigue siendo la linea
   1",a$
```

Ejecute este programa (basta con que pulse [INTRO] cada vez que el programa se detenga). Cuando el programa escribe **Esta es la linea 2**, la pantalla inferior se desplaza hacia arriba para dejarle sitio; pero también la numeración se desplaza hacia arriba, de forma que las líneas del texto conservan los mismos números.

Otro refinamiento de la sentencia **INPUT** que no hemos visto todavía es la opción **LINE**, que nos ofrece otra forma de captar por el teclado los valores de las variables literales. Si ponemos **LINE** antes del nombre de la variable literal que va a ser captada, como por ejemplo en

```
INPUT LINE a$
```

el **+2A** no mostrará las comillas como de costumbre (aunque para sus adentros actuará como si las hubiera escrito). Entonces, si respondemos escribiendo

gato

INPUT asignará el valor **gato** a **a$**. Puesto que las comillas no aparecen rodeando a la cadena, no podremos borrarlas. Tenga en cuenta que no se puede usar **LINE** cuando la variable es numérica.

Hay un interesante efecto secundario de **INPUT**. Mientras se está respondiendo a una sentencia **INPUT**, el antiguo sistema Spectrum de introducción de palabras clave por la pulsación de una sola tecla se comporta de una forma extraña, hasta que lo hacemos entrar en vereda al pulsar [INTRO]. Ejecute este programa:

```
10 INPUT numeros
20 PRINT numeros
30 GO TO 10
```

Introduzca unos cuantos números y verá cómo éstos son reproducidos fielmente en la pantalla. Después pulse la tecla [EXTRA] seguida de [M]. Aparece la palabra clave **PI**; si ahora pulsa [INTRO], el **+2A** escribe 3.1415927 como por arte de magia. Sin embargo, si usted

escribe las letras PI sin pulsar antes [EXTRA] y [M], el **+2A** detiene el programa y emite el informe

2 VARIABLE NO DEFINIDA

No hay una explicación sencilla para este comportamiento. En todo caso, conviene saber que cosas de éstas pueden suceder si pulsamos ciertas combinaciones de teclas en respuesta a **INPUT**.

Los caracteres de control **CHR$ 22** y **CHR$ 23** producen unos efectos similares a los de **AT** y **TAB**. Siempre que se le pide al **+2A** que «escriba» uno de ellos, el carácter debe ir seguido por dos caracteres más, que son tratados por el ordenador como parámetros de **AT** o de **TAB**. Normalmente es más cómodo usar explícitamente **AT** y **TAB** que estos dos caracteres, aunque hay situaciones en que pueden ser útiles.

El carácter de control que corresponde a **AT** es **CHR$ 22**. El primer número que se especifica a continuación es el número de fila; el siguiente, el número de columna. Así,

```
PRINT CHR$ 22+ CHR$ 1+ CHR$ c;
```

equivale exactamente a

```
PRINT AT 1, c;
```

Esto es así a pesar de que **CHR$ 1** o **CHR$ c** tendrían en otra situación significados diferentes (por ejemplo, cuando **c** es 13); el **CHR$ 22** que los precede les cambia el significado.

El carácter de control que corresponde a **TAB** es **CHR$ 23**; los dos números que le siguen se combinan para dar un número del margen de 0 a 65535, que es el que se toma como parámetro de **TAB**. La sentencia

```
PRINT CHR$ 23+ CHR$ a+ CHR$ b;
```

es equivalente a

```
PRINT TAB a+256*b;
```

Con **POKE** podemos hacer que el ordenador deje de preguntarnos si queremos desplazar la pantalla ('¿**MAS?**'). Para ello podemos escribir

```
POKE 23692, 255
```

de vez en cuando. Después de hacer esto, la pantalla tendrá que desplazarse 255 veces antes de que el **+2A** nos haga la pregunta '¿**MAS?**'. Como ejemplo, pruebe

```
10 FOR n=0 TO 1000
20 PRINT n: POKE 23692, 255
30 NEXT n
```

y observe cómo los números se escapan de la pantalla.

Ejercicio

1. Pruebe este programa con algún niño para ver si se sabe las tablas de multiplicar:

```
 10 LET m$=""
 20 LET a=INT ( RND *10)+1: LET b=INT ( RND *10)+1
 30 INPUT (m$) ' ' "¿Cuanto es ";(a);" por ";(b);"?";c
100 IF c=a*b THEN LET m$="¡Bien!": GO TO 20
110 LET m$="Mal. Prueba otra vez.": GO TO 30
```

   Si el niño es un poco despabilado, puede darse cuenta de que no necesita hacer el cálculo él mismo. Por ejemplo, si el **+2A** le pregunta que cuánto es 2 por 3, todo lo que tiene que hacer es escribir **2*3** literalmente.

Sección 16
Colores

Temas tratados

INK, PAPER, FLASH, BRIGHT, INVERSE, OVER, BORDER

Pruebe este programa

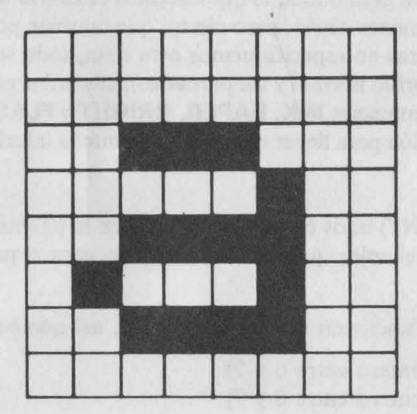
```
 10 FOR m=0 TO 1: BRIGHT m
 20 FOR n=1 TO 10
 30 FOR c=0 TO 7
 40 PAPER c: PRINT "    ";: REM 4 espacios de colores
 50 NEXT c: NEXT n: NEXT m
 60 FOR m=0 TO 1: BRIGHT m: PAPER 7
 70 FOR c=0 TO 3
 80 INK c: PRINT c;"  ";: REM 2 espacios
 90 NEXT c: PAPER 0
100 FOR c=4 TO 7
110 INK c: PRINT c;"  ";: REM 2 espacios
120 NEXT c: NEXT m
130 PAPER 7: INK 0: BRIGHT 0
```

Esto muestra los ocho colores (incluidos el blanco y el negro) y los dos niveles de brillo que puede producir el **+2A** en un televisor de color. (Si su aparato es de blanco y negro, lo que podrá apreciar es diversas intensidades de gris.) Una forma más rápida, y drástica, de obtener el mismo resultado consiste en reinicializar (botón **REINIC**) el **+2A** mientras se tiene pulsada la tecla [BREAK]. Veamos la lista de los colores y sus códigos:

0 negro
1 azul
2 rojo
3 magenta
4 verde
5 cyan
6 amarillo
7 blanco

En los televisores de blanco y negro, estos números están por orden de luminosidad creciente. Para usar los colores adecuadamente es necesario entender cómo se forma la imagen en la pantalla.

La imagen está dividida en 768 posiciones (24 filas por 32 columnas), llamadas *celdas*, en las que podemos escribir los caracteres.



Una celda de carácter típica

Cada celda de carácter consiste en una retícula de 8×8 puntos (como la de la figura anterior). Esto debería recordarle a usted los gráficos de usuario de la Sección 14, donde representábamos con un 0 los puntos blancos y con un 1 los negros.

El carácter tiene dos colores asociados con él: la *tinta*, o color del primer plano, que es el color con el que el **+2A** dibuja los puntos negros de nuestro cuadrado, y el *papel*, o color del fondo, que es el usado para los puntos blancos. Inicialmente todas las celdas tienen tinta negra y papel blanco, de forma que los caracteres aparecen en negro sobre blanco.

El carácter también tiene un nivel de brillo o luminosidad (normal o extra) asociado, y algo que indica si parpadea o no. El parpadeo consiste en un intercambio continuo de los colores de la tinta y el papel. Toda esta información puede ser codificada en números, de modo que un carácter tiene lo siguiente:

(i) Una retícula de ceros y unos que definen la forma del carácter (0 para el papel, 1 para la tinta).

(ii) Un código para el color de la tinta y otro para el del papel, cada uno codificado mediante un número del 0 al 7.

(iii) Un código para el brillo (0 para el normal, 1 para el extra).

(iv) Un código para el parpadeo (0 para constante, 1 para parpadeo).

Puesto que cada código de color de tinta y de papel se refiere a una celda de carácter completa, es imposible tener más de dos colores en un bloque dado de 64 puntos. Lo mismo ocurre con los códigos de brillo y parpadeo: se refieren a la celda de carácter completa,

no a puntos individuales dentro de ella. Los códigos de color, brillo y parpadeo de un carácter dado son lo que denominamos *atributos* de ese carácter.

Cuando escribimos algo en la pantalla, lo que hacemos es alterar la situación de los puntos de la celda afectada. Es menos obvio, pero cierto, que también podemos cambiar los atributos de la celda. Mientras no especifiquemos otra cosa, todo se escribe con tinta negra sobre papel blanco (con brillo normal y sin parpadeo); sin embargo, podemos cambiar esta situación utilizando las sentencias **INK, PAPER, BRIGHT** y **FLASH**. Seleccione la opción **Pantalla** del menú de edición para llevar el cursor a la pantalla inferior; dé la siguiente orden:

```
PAPER 5
```

y luego escriba (con **PRINT**) unos cuantos caracteres en la pantalla; aparecerán sobre papel cyan, porque éste es el color que habíamos elegido para el papel (el código del color cyan es el 5).

Las otras instrucciones funcionan de manera similar, así que con

PAPER (número entero entre 0 y 7)
INK (número entero entre 0 y 7)
BRIGHT (número entero entre 0 y 1)
FLASH (número entero entre 0 y 1)

podemos controlar todos los atributos de los caracteres que escribamos a continuación.

Pruebe todas estas instrucciones. Cuando lo haya hecho ya podrá entender cómo funcionaba el programa del principio de esta sección (recuerde que un espacio es un carácter en el que todos los puntos tienen el color del papel).

Hay otros números que podemos usar en estas sentencias y cuyos efectos son menos directos.

El **8** se puede usar en todas las sentencias y significa 'transparente', en el sentido de que respeta el atributo que estuviera antes en vigor. Supongamos, por ejemplo, que damos la orden

```
PAPER 8
```

Ninguna posición de carácter puede tener este color de papel, sencillamente porque no existe el color 8; lo que ocurre es que cuando escribamos sobre una posición, el color del papel seguirá siendo el que hubiera antes en ella. Sin embargo, **INK 8**, **BRIGHT 8** y **FLASH 8** funcionan del mismo modo que los otros números de atributo.

El número **9** sólo es aplicable a **PAPER** e **INK** y significa 'contraste'. Sirve para hacer que el color (de la tinta o del papel) utilizado contraste con el otro, convirtiéndolo en blanco si el otro es un color oscuro (negro, rojo o magenta) o haciéndolo negro si el otro es claro (verde, cyan, amarillo o blanco).

Compruébelo dando estas instrucciones:

```
INK 9: FOR c=0 TO 7: PAPER c: PRINT c: NEXT c
```

Una muestra más impresionante de su potencia es ejecutar el programa del principio para que dibuje las rayas de colores (de nuevo, seleccione la pantalla inferior antes de escribir **RUN**) y luego hacer lo siguiente:

```
INK 9: PAPER 8: PRINT AT 0,0;: FOR n=1 TO 1000: PRINT n;: NEXT n
```

El color de la tinta contrasta siempre con el color antiguo del papel en todas las celdas de carácter.

Los televisores de color en realidad sólo utilizan tres colores: rojo, verde y azul. Todos los demás colores son mezclas de éstos. Por ejemplo, el magenta se forma mezclando rojo con azul; ésta es la razón por la que su código es 3, ya que este número es la suma de los códigos del rojo (2) y el azul (1).

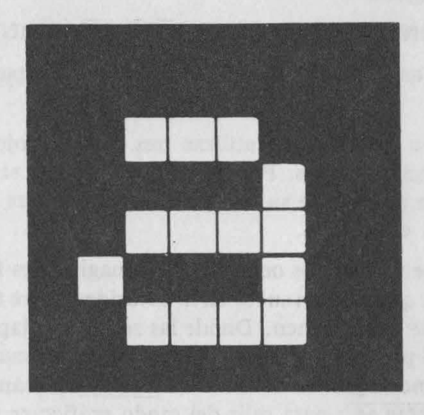
Para comprender cómo se forman los ocho colores, imagine tres focos rectangulares, de colores rojo, verde y azul, que arrojan su luz en la oscuridad sobre tres zonas parcialmente solapadas de un pedazo de papel blanco. Donde las zonas se solapen veremos mezclas de colores, como muestra el programa siguiente (observe que los cuadrados de tinta sólida se obtienen activando el modo gráfico con la tecla GRAF y pulsando luego la tecla 8 en combinación con MAYUSCULAS; para salir del modo gráfico se pulsa la tecla 9):

```
  10 BORDER 0: PAPER 0: INK 7: CLS
  20 FOR a=1 TO 6
  30 PRINT TAB 6; INK 1;"                  ": REM 18 cuadrados de tinta
  40 NEXT a
  50 LET linea datos=200
  60 GO SUB 1000
  70 LET linea datos=210
  80 GO SUB 1000
  90 STOP
 200 DATA 2,3,7,5,4
 210 DATA 2,2,6,4,4
1000 FOR a=1 TO 6
1010 RESTORE linea datos
1020 FOR b=1 TO 5
1030 READ c: PRINT INK c;"      ";: REM 6 cuadrados de tinta
1040 NEXT b: PRINT: NEXT a
1050 RETURN
```

Hay una función, llamada **ATTR** que averigua los atributos de una posición dada de la pantalla. Es una función complicada, y por eso la aplazaremos para el final de esta sección.

Disponemos de otras dos sentencias, **INVERSE** y **OVER**, que no controlan los atributos, sino la distribución de puntos del carácter que resulta impreso en la pantalla. Llevan como parámetro el número 0, que significa 'activado', o el número 1, que significa 'desactivado'. Así, **INVERSE 1** invierte la situación de todos los puntos de la retícula: apaga

los encendidos y enciende los apagados. Por ejemplo, la celda de carácter de la letra 'a' se convierte en la que se muestra en la siguiente figura:



Si, como ocurre en el momento de encender el ordenador, tenemos tinta negra y papel blanco, la 'a' aparecerá en blanco sobre negro.

La sentencia

OVER 1

activa un tipo particular de sobreimpresión. Normalmente, cuando se escribe algo en una posición de la pantalla, el carácter nuevo «tapa» completamente lo que hubiera antes en ella; en cambio, si previamente se da la orden **OVER 1**, el nuevo carácter se funde con el antiguo. Esto permite escribir caracteres compuestos, por ejemplo, letras subrayadas, como en el programa siguiente. (Reinicialice el ordenador y seleccione **+3 BASIC**. El carácter de subrayado se obtiene con [SIMB] y [0].)

```
10 OVER 1
20 PRINT "w"; CHR$ 8;"_";
```

(Observe que hemos usado el carácter de control **CHR$ 8** para hacer retroceder la posición de escritura antes de escribir el '_'.)

Hay otra forma de usar **INK**, **PAPER**, etc. que usted probablemente encontrará más cómoda que la que hemos visto hasta ahora. Podemos utilizarlas como elementos de **PRINT** seguidas del signo de punto y coma, y harán exactamente lo mismo que si las hubiésemos ejecutado como sentencias independientes, con la única diferencia de que sus efectos son sólo temporales, pues duran solamente hasta el final de la sentencia **PRINT** que las contiene. Por tanto, si escribimos

```
PRINT PAPER 6;"x";: PRINT "y"
```

sólo la **x** quedará escrita sobre papel amarillo.

Cuando usamos **INK** y sus compañeras como sentencias independientes, no afectan al color de la pantalla inferior, en la que se realiza la captación de datos por las sentencias **INPUT** y donde el **+2A** muestra los mensajes de error. En esta parte de la pantalla, el color del papel es el del borde, el de la tinta es el 9 (contraste), carece de parpadeo y su brillo es normal. Como color del borde se puede elegir cualquiera de los ocho colores normales (no el 8 ni el 9) mediante la instrucción

BORDER *color*

Cuando se está ejecutando **INPUT**, los caracteres introducidos aparecen en tinta de contraste sobre papel del mismo color que el borde; pero se puede cambiar el color de los mensajes inductores escritos por el **+2A** usando para ello **INK** y **PAPER** (y las demás) como elementos de **INPUT**, de la misma forma que se haría en una sentencia **PRINT**. Sus efectos duran hasta el final de la sentencia o hasta que se termine de introducir los datos, lo que antes ocurra. Pruebe lo siguiente:

```
INPUT FLASH 1; INK 1;"¿Cual es su numero?";n
```

El **+2A** tiene en gran estima su salud mental: cualquiera que sea la combinación de colores producida por el programa, el editor siempre funciona con tinta negra sobre papel blanco.

Otra forma de cambiar los colores consiste en usar caracteres de control, de forma parecida a como hacíamos con **AT** y **TAB** en la Sección 15.

CHR$ 16	corresponde a	**INK**
CHR$ 17	corresponde a	**PAPER**
CHR$ 18	corresponde a	**FLASH**
CHR$ 19	corresponde a	**BRIGHT**
CHR$ 20	corresponde a	**INVERSE**
CHR$ 21	corresponde a	**OVER**

Estos caracteres de control van seguidos de un carácter que especifica un color; así, por ejemplo,

```
PRINT CHR$ 16+ CHR$ 9;"elemento"
```

produce el mismo efecto que

```
PRINT INK 9;"elemento"
```

En la práctica, no tiene ningún interés utilizar estos caracteres de control, pues siempre podemos emplear en su lugar las sentencias **INK**, **PAPER**, etc. Sin embargo, si usted tiene en cassette algún programa escrito para el antiguo 48 BASIC, puede encontrar tales caracteres de control inmersos en el listado. En general, el editor los ignorará activamente y los suprimirá a la primera oportunidad. No es posible introducirlos en los listados, como se hacía en el antiguo Spectrum de 48K.

La función **ATTR** tiene la forma

ATTR (*fila,columna*)

Sus dos argumentos son los números de fila y columna que ya sabemos utilizar en la cláusula **AT**. El resultado de la función es un número que informa sobre los colores y los demás atributos de la posición de carácter especificada. Esta función puede intervenir en expresiones, con la misma libertad que cualquier otra función.

El número resultante es una combinación de cuatro números que se construye de la siguiente forma:

sumar	128	si la celda de carácter está parpadeando; 0 en otro caso
sumar	64	si la celda de carácter es brillante; 0 si es normal
sumar	8	multiplicado por el código del color del papel
sumar	1	multiplicado por el código del color de la tinta

Por ejemplo, si la celda de carácter está parpadeando, tiene un brillo normal, papel amarillo y tinta azul, los cuatro números que debemos sumar son 128, 0, 8*6=48 y 1, lo que da un total de 177. Compruébelo escribiendo:

```
PRINT AT 0,0; FLASH 1; PAPER 6; INK 1;" ";ATTR (0,0)
```

Ejercicios

1. Pruebe la siguiente orden:

```
PRINT "–"; CHR$ 8; OVER 1;"/";
```

En el lugar de intersección de la barra con el signo menos ha quedado un punto blanco. Ésta es la forma de sobreimpresión en el **+2A**: papel más papel y tinta más tinta dan papel; tinta más papel da tinta. Este sistema tiene la interesante propiedad de que si sobreescribimos la misma cosa dos veces, se restaura el carácter original. Sabido esto, si ahora hacemos

```
PRINT CHR$ 8; OVER 1;"/"
```

¿por qué recuperamos un '–' incólume?

2. Ejecute este programa:

```
10 POKE 225227+ RND *704, RND *127
20 GO TO 10
```

(No se preocupe por cómo funciona el programa.) El programa está cambiando los colores de las celdas de la pantalla; **RND** garantiza que esto ocurre de forma aleatoria. Las rayas diagonales que finalmente llegan a aparecer son la manifestación del hecho de que **RND** no genera verdaderos números aleatorios, sino pseudoaleatorios.

Sección 17
Gráficos

Temas tratados:

PLOT, DRAW, CIRCLE
Pixels

A lo largo de toda esta sección, es conveniente que escriba los programas de ejemplo, las órdenes y la palabra **RUN** en la pantalla inferior (selecciónela con la opción **Pantalla** del menú de edición).

En esta sección vamos a aprender a dibujar figuras en el **+2A**. La parte de la pantalla que podemos utilizar para los dibujos tiene 22 filas y 32 columnas, lo que da un total de 22*32=704 posiciones de carácter. Como recordará de la Sección 16, cada posición de carácter es una retícula de 8×8 puntos, llamados *pixels* (de *picture elements*, 'elementos de imagen').

Un pixel se especifica dando dos números: sus *coordenadas*. El primero, su coordenada *x*, indica a qué distancia se encuentra el pixel del borde izquierdo de la pantalla. El segundo, su coordenada *y*, indica la distancia que hay del pixel al borde inferior. Se suele escribir las coordenadas entre paréntesis y separadas por comas, de modo que, por ejemplo, (0,0), (255,0), (0,175) y (255,175) representan, respectivamente, los rincones inferior izquierdo, inferior derecho, superior izquierdo y superior derecho de la pantalla.

Si no consigue recordar cuál es la coordenada *x* y cuál la coordenada *y*, quizá le sirva de ayuda asociar mentalmente la forma de la 'y' con la 'v' de 'vertical'.

La sentencia

PLOT *coordenada-x, coordenada-y*

«tiñe» el pixel correspondiente a esas coordenadas, de forma que este programa

```
10 PLOT INT ( RND *256), INT ( RND *176): INPUT a$: GO TO 10
```

dibuja un punto en una posición aleatoria cada vez que pulsamos INTRO.

Veamos un programa más interesante que traza un gráfico de la función **SIN** (una onda sinusoidal) para valores del arco comprendidos entre 0 y 2π:

```
10 FOR n=0 TO 255
20 PLOT n, 88+80* SIN (n/128* PI )
30 NEXT n
```

El siguiente programa traza un gráfico de **SQR** (parte de una parábola) entre 0 y 4:

```
10 FOR n=0 TO 255
20 PLOT n, 80* SQR (n/64)
30 NEXT n
```

Observe que las coordenadas que describen la posición de los pixels son distintas de las que utilizábamos para dar el número de fila y de columna en la cláusula **AT**. Quizá le resulte útil el diagrama de la Sección 15 de este capítulo a la hora de elegir las coordenadas de los pixels y los números de fila y columna.

PLOT realiza los dibujos punto a punto. También podemos dibujar rectas, circunferencias y trozos de circunferencia utilizando las sentencias **DRAW** y **CIRCLE**.

La sentencia **DRAW** dibuja una recta; su forma es:

DRAW *x,y*

El punto de partida de la recta es el pixel en el que se detuvo la última sentencia **PLOT**, **DRAW** o **CIRCLE** (que es lo que llamamos *posición actual* del cursor gráfico; **RUN**, **CLEAR**, **CLS** y **NEW** la restablecen en el extremo inferior izquierdo, punto 0,0). El punto final de la recta se encuentra a *x* pixels a la derecha de la posición actual y a *y* por arriba. La sentencia **DRAW** determina la longitud y la dirección de la recta, pero no su punto de partida.

Experimente con unas cuantas órdenes **PLOT** y **DRAW**; por ejemplo,

```
PLOT 0,100: DRAW 80,-35
PLOT 90,150: DRAW 80,-35
```

Observe que los números de la sentencia **DRAW** pueden ser negativos, porque representan desplazamientos con respecto a una posición, mientras que los de **PLOT** tienen que ser positivos, porque representan coordenadas absolutas.

También podemos dibujar en color, aunque hay que tener presente que los colores cubren siempre la totalidad de la celda de un carácter y no pueden ser especificados para pixels individuales. Cuando dibujamos un pixel, éste se tiñe del color de la tinta, y el resto de la celda en que se encuentra el pixel toma el mismo color, como demuestra el siguiente programa:

```
10 BORDER 0: PAPER 0: INK 7: CLS: REM tiñe la pantalla de negro
20 LET x1=0: LET y1=0: REM comienzo de la recta
30 LET c=1: REM para color de tinta, comenzando con azul
40 LET x2= INT ( RND *256): LET y2= INT ( RND *176): REM final aleatorio
   de la recta
50 DRAW INK c;x2-x1,y2-y1
60 LET x1=x2: LET y1=y2: REM la siguiente recta comienza donde acabó la
   anterior
70 LET c=c+1: IF c=8 THEN LET c=1: REM otro color
80 GO TO 40
```

Las rectas parecen hacerse más amplias a medida que avanza el programa. Esto se debe a que cada recta cambia los colores de todas las celdas que atraviesa. Observe que podemos introducir las cláusulas **PAPER**, **INK**, **FLASH**, **BRIGHT**, **INVERSE** y **OVER** en las sentencias **PLOT** y **DRAW**, lo mismo que hacíamos en **PRINT** e **INPUT**. Estas cláusulas van entre la palabra clave y las coordenadas y terminan en coma o en punto y coma.

Una característica adicional de **DRAW** es que podemos usarla para dibujar fragmentos de circunferencias en vez de rectas. La sentencia necesita entonces un número más para especificar el ángulo del trozo de circunferencia. La forma es

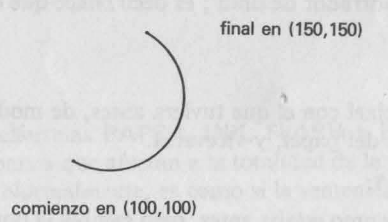
DRAW *x,y,a*

donde *x* e *y* especifican, al igual que antes, el punto final de la línea, y donde *a* especifica el arco (en número de radianes). Si *a* es positivo, el giro se realiza hacia la izquierda; si es negativo, hacia la derecha. Otra forma de entender el significado de *a* es considerar que indica qué fracción de una circunferencia completa se va a dibujar (una circunferencia completa tiene 2π radianes). Por ejemplo, si *a* es π, obtendremos una semicircunferencia; si *a* es 0.5π, un cuarto de circunferencia; y así sucesivamente.

Supongamos que *a* vale π. Entonces, cualesquiera que sean los valores de *x* e *y*, siempre obtendremos una circunferencia. Pruebe el programa:

```
10 PLOT 100,100: DRAW 50,50,PI
```

que dibuja lo siguiente:



El dibujo comienza en dirección sudeste, pero, en el momento en que se detiene, ya va en dirección noroeste. Entre el rumbo inicial y el final ha girado 180 grados, o π radianes, que es el valor de *a*.

Ejecute el programa varias veces reemplazando **PI** por otras expresiones; por ejemplo: **−PI, PI/2, 3*PI/2, PI/4, 1, 0**, etc.

La última sentencia de esta sección es **CIRCLE**, que dibuja una circunferencia completa. La circunferencia se especifica dando las coordenadas del centro y el radio. La forma de **CIRCLE** es:

CIRCLE *coordenada-x,coordenada-y,radio*

Al igual que sucedía con **PLOT** y **DRAW**, al principio de **CIRCLE** podemos poner cláusulas que controlen el color y los otros atributos del dibujo.

La función **POINT** averigua si un pixel tiene el color de la tinta o el del papel. Sus dos argumentos son las coordenadas del pixel (que deben ir entre paréntesis). El resultado es 0 si el pixel tiene el color del papel, o 1 si es del color de la tinta. Pruebe lo siguiente:

CLS : PRINT POINT (0,0): PLOT 0,0: PRINT POINT (0,0)

Escriba:

PAPER 7: INK 0

e investiguemos ahora cómo funcionan **INVERSE** y **OVER** dentro de una sentencia **PLOT**. Ambas afectan sólo al pixel especificado en **PLOT**, no al resto de la celda de carácter. Normalmente estas cláusulas están desactivadas (a 0) en las sentencias **PLOT**, así que sólo es necesario mencionarlas cuando se desee activarlas (ponerlas a 1).

He aquí una lista de las posibilidades:

PLOT;

Ésta es la forma usual. Dibuja un punto con la tinta; es decir, tiñe el pixel con el color de la tinta.

PLOT INVERSE 1;

Dibuja un punto con el 'borrador de tinta'; es decir, hace que el pixel muestre el color del papel.

PLOT OVER 1;

Intercambia el color del pixel con el que tuviera antes, de modo que si antes era el de la tinta lo convierte en el del papel, y viceversa.

PLOT INVERSE 1; OVER 1;

Deja el pixel exactamente como estaba antes, pero cambia la posición actual; puede servir, pues, para trasladar el cursor gráfico.

Para ver otro ejemplo de aplicación de la sentencia **OVER**, llene la pantalla de texto escrito en negro sobre blanco y después dé la orden:

PLOT 0,0: DRAW OVER 1;255,175

Esto dibuja una recta bastante aceptable, aunque con puntos en blanco cada vez que se tropieza con algo escrito previamente. Ahora dé otra vez exactamente la misma orden: la recta desaparece sin dejar rastro alguno; ésta es la gran ventaja de **OVER 1**. Si hubiéramos dibujado la recta con

PLOT 0,0: DRAW 255,175

y la hubiéramos borrado con

PLOT 0,0: DRAW INVERSE 1;255,175

habríamos borrado también parte del texto.

Ahora pruebe

PLOT 0,0: DRAW OVER 1;250,175

y trate de borrar con

DRAW OVER 1;-250,-175

Esto no funciona demasiado bien, y es que los pixels por los que pasa la recta en su camino de vuelta no son exactamente los mismos que los que había recorrido a la ida. Por consiguiente, para borrar una recta hay que recorrerla en el mismo sentido en que se la dibujó.

Una manera de obtener colores especiales es mezclar dos colores normales en un cuadrado, usando para ello un gráfico de usuario. Pruebe el siguiente programa:

```
1000 FOR n=0 TO 6 STEP 2
1010 POKE USR "a"+n, BIN 01010101: POKE USR "a"+n+1, BIN 10101010
1020 NEXT n
```

que genera un gráfico de usuario con el diseño de un tablero de ajedrez. Si ahora escribimos este carácter (pulse [GRAF] y luego [A]) en tinta roja sobre papel amarillo, obtendremos un naranja bastante aceptable.

Ejercicios

1. Experimente con las cláusulas **PAPER, INK, FLASH** y **BRIGHT** en una sentencia **PLOT**. Éstas son las partes que afectan a la totalidad de la celda de carácter en la que se encuentra el pixel. Normalmente, es como si la sentencia **PLOT** hubiera empezado con

   PLOT PAPER 8; FLASH 8; BRIGHT 8; etc.

   y sólo se altera el color de la tinta de una celda de carácter cuando se dibuja algo en ella, pero usted puede cambiar esta situación si lo desea.

   Tenga cuidado especialmente cuando use colores con **INVERSE 1**, ya que esta cláusula hace que el pixel muestre el color del papel, y puede cambiar el color de la tinta, el cual pudiera no ser el que usted esperaba.

2. Pruebe a dibujar circunferencias usando **SIN** y **COS** con el siguiente programa (si ha leído la Sección 10, trate de entender cómo funciona):

```
10 FOR n=0 TO 2* PI STEP PI /180
20 PLOT 100+80* COS n, 87+80* SIN n
30 NEXT n
40 CIRCLE 150, 87, 80
```

Como puede ver, la sentencia **CIRCLE** es mucho más rápida, pero menos precisa.

3. Pruebe lo siguiente:

```
CIRCLE 100,87,80: DRAW 50,50
```

De esto se deduce que la sentencia **CIRCLE** deja el cursor en una posición bastante imprecisa (siempre se trata de algún punto a media altura en el lado derecho de la circunferencia). Normalmente habrá que ejecutar una sentencia **PLOT** a continuación de una **CIRCLE** antes de seguir dibujando.

Sección 18
Control del tiempo

Temas tratados:

PAUSE, PEEK, INKEY$

Con frecuencia se necesita poder controlar el tiempo que dura la ejecución de un programa. Para este fin se dispone de la sentencia **PAUSE**.

PAUSE *n*

detiene el programa y mantiene la imagen durante *n* barridos del cuadro (a razón de 50 barridos por segundo). El valor de *n* puede llegar hasta 65535, número que produce una pausa de 22 minutos. Si *n*=0, la pausa es de duración infinita.

Para abandonar la pausa en cualquier momento se pulsa una tecla.

El siguiente programa mueve el segundero de un reloj:

```
 10 REM primero dibujamos la esfera del reloj
 20 FOR n=1 TO 12
 30 PRINT AT 10-10* COS (n/6* PI ),16+10* SIN (n/6* PI );n
 40 NEXT n
 50 REM ahora ponemos en marcha el reloj
 60 FOR t=0 TO 200000: REM t es el tiempo en segundos
 70 LET a=t/30* PI : REM a es el ángulo del segundero en radianes
 80 LET sx=80* SIN a: LET sy=80* COS a
200 PLOT 128,88: DRAW OVER 1;sx,sy: REM dibuja el segundero
210 PAUSE 42
220 PLOT 128,88: DRAW OVER 1;sx,sy: REM borra el segundero
400 NEXT t
```

El reloj deja de funcionar al cabo de unas 55.5 horas, debido a la línea 60, pero usted puede darle más cuerda fácilmente. Observe que el control del tiempo lo lleva la línea 210. Parecería natural que **PAUSE 50** realizase el batido de segundos, pero también hay que tener en cuenta el tiempo que tarda el ordenador en llevar a cabo las restantes instrucciones del programa. La forma de ajustar el parámetro de **PAUSE** es comparar el funcionamiento del programa con un cronómetro hasta dar con el valor correcto. (El ajuste no puede ser demasiado perfecto; una diferencia de un barrido más o menos representa un error del 2%, es decir, aproximadamente media hora al día.)

Hay otro método mucho más exacto para medir el tiempo, basado en el contenido de ciertas posiciones de memoria. Para leer los datos almacenados en la memoria se utiliza la función **PEEK**. La Sección 25 de este capítulo explica con detalle el significado exacto de las posiciones de memoria que vamos a leer. La expresión que necesitamos es:

```
(65536* PEEK 23674+256* PEEK 23673+ PEEK 23672)/50
```

que da el número de segundos transcurridos desde que se encendió o reinicializó el ordenador (hasta alrededor de tres días y 21 horas, porque después de ese tiempo vuelve a 0).

El siguiente programa es una versión revisada del programa anterior en la que utilizamos esa expresión:

```
10 REM primero dibujamos la esfera del reloj
20 FOR n=1 TO 12
30 PRINT AT 10-10* COS (n/6* PI ),16+10* SIN (n/6* PI );n
40 NEXT n
50 DEF FN t()= INT ((65536* PEEK 23674+256* PEEK 23673+ PEEK 23672)/50):
   REM número de segundos desde la reinicialización
100 REM ahora ponemos en marcha el reloj
110 LET t1= FN t()
120 LET a=t1/30* PI: REM a es el ángulo del segundero en radianes
130 LET sx=72* SIN a: LET sy=72* COS a
140 PLOT 131,91: DRAW OVER 1;sx,sy: REM dibujar segundero
200 LET t= FN t()
210 IF t<=t1 THEN GO TO 200: REM esperar hasta que llegue el momento de
   moverlo
220 PLOT 131,91: DRAW OVER 1;sx,sy: REM borrar segundero
230 LET t1=t: GO TO 120
```

El reloj interno en el que se basa este método tiene una precisión de alrededor del 0.01% (aproximadamente 10 segundos por día), a condición de que lo único que haga el ordenador sea ejecutar el programa. Sin embargo, cuando se ejecuta la sentencia **BEEP** (descrita en la Sección 19 de este capítulo) o se trabaja con el magnetófono o con cualquier periférico (por ejemplo, la impresora o la unidad de disco externa), el reloj interno se detiene temporalmente, y por lo tanto se atrasa.

Los números **PEEK 23674**, **PEEK 23673** y **PEEK 23672** están almacenados en el ordenador y constituyen un contador de cincuentavos de segundo. Crecen gradualmente de 0 a 255, y, al llegar a 255, vuelven a cero.

El que se incrementa más a menudo es **PEEK 23672** (una unidad cada 1/50 segundos). Cuando llega a 255, el siguiente incremento lo lleva a 0, y al mismo tiempo suma 1 a **PEEK 23673**. Cuando **PEEK 23673** pasa de 255 a 0 (cada 256/50 segundos), esto a su vez suma 1 a **PEEK 23674**. Esta explicación debería ser suficiente para que usted comprenda cómo funciona la expresión anterior.

Supongamos que nuestros tres números son 0 (**PEEK 23674**), 255 (**PEEK 23673**) y 255 (**PEEK 23672**). Esto significa que hace unos 21 minutos que se encendió el ordenador. El valor de la expresión será (65536*0+256*255+255)/50=1310.7.

Pero hay un peligro oculto: la próxima vez que se incremente el contador en 1/50 segundos, los tres números cambiarán a 1, 0 y 0. Alguna vez ocurrirá esto en el preciso momento en que el ordenador está calculando la expresión; entonces el **+2A** leerá el número 0 en **PEEK 23674**, pero los otros dos valores cambiarán a cero antes de que el programa los haya leído. El valor de la expresión será en ese caso (65536*0+256*0+0)/50=0, que evidentemente no es correcto.

Una manera sencilla de evitar este problema es evaluar la expresión dos veces seguidas y tomar la respuesta mayor. Si observa cuidadosamente el programa anterior, verá que hace esto implícitamente.

Veamos un truco para aplicar esta regla. Defina las funciones:

```
10 DEF FN m(x,y)=(x+y+ ABS (x−y))/2: REM el mayor de x e y
20 DEF FN u()=(65536* PEEK 23674+256* PEEK 23673+ PEEK 23672)/50:
   REM tiempo (puede ser incorrecto)
30 DEF FN t()= FN m( FN u(), FN u()): REM tiempo (correcto)
```

Podemos cambiar los tres números del contador para que den el tiempo real en vez del tiempo que lleva el ordenador encendido. Por ejemplo, para poner el contador en hora a las 10 de la mañana, observemos que 10 horas son 10×60×60×50=1800000 cincuentavos de segundo y que 1800000=65536*27+256*119+64.

Por eso debemos escribir los números 27, 119 y 64 en las tres posiciones de memoria, y eso se hace con la siguiente orden:

```
POKE 23674,27: POKE 23673,119: POKE 23672,64
```

En los países en los que la frecuencia de la red de energía eléctrica sea de 60 Hz, el número 50 que aparece en todos estos programas debe ser cambiado por 60.

La función **INKEY$** (que no lleva argumento) lee el teclado. Si en el momento de evaluar **INKEY$** tenemos pulsada una tecla sola (o combinada con **[MAYUSCULAS]**), el resultado es el carácter que esa tecla da normalmente; de lo contrario, el resultado es la cadena vacía.

Pruebe este programa, que funciona como una máquina de escribir:

```
10 IF INKEY$ <> "" THEN GO TO 10
20 IF INKEY$ ="" THEN GO TO 20
30 PRINT INKEY$ ;
40 GO TO 10
```

La línea 10 espera hasta que usted retira sus dedos del teclado; la línea 20 espera hasta que pulsa una nueva tecla.

Recuerde que, a diferencia de **INPUT**, **INKEY$** no le espera, de modo que no tiene que pulsar [INTRO]. Por otra parte, si usted no escribe absolutamente nada, ha perdido su oportunidad.

Ejercicios

1. ¿Qué ocurre si omitimos la línea 10 del programa de la máquina de escribir?

2. Otra forma de usar **INKEY$** consiste en combinarla con **PAUSE**, como en este segundo programa de máquina escribir:

```
10 PAUSE 0
20 PRINT INKEY$ ;
30 GO TO 10
```

   ¿Por qué es esencial, para que esto funcione, que la pausa no acabe si ya se encuentra usted pulsando una letra cuando comienza?

3. Adapte el programa del segundero de forma que también muestre el minutero y la manecilla de las horas, redibujándolas una vez por minuto. Si se siente ambicioso, haga que produzca algún efecto adicional cada cuarto de hora, quizá el repique de las campanas del 'Big Ben' usando **PLAY** (orden descrita en la Sección 19 de este capítulo).

Sección 19
Sonido

Temas tratados:

BEEP, PLAY

Como usted ya habrá observado, el **+2A** puede producir diversos ruidos. Para obtener la mejor calidad de sonido posible, es importante que el televisor esté perfectamente sintonizado (v. Capítulo 2). Si en lugar de un televisor está usted usando un monitor (que no reproducirá el sonido del **+2A**), observe que puede tomar del zócalo **CINTA/SONIDO** de la parte posterior del ordenador una señal de sonido, la cual puede ser enviada a los altavoces o auriculares a través de un amplificador de audio. Tenga en cuenta, no obstante, que los auriculares *no* pueden ser conectados directamente al ordenador.

Las conexiones del zócalo **CINTA/SONIDO** están descritas en el Capítulo 10.

El conocimiento de la terminología musical ayuda a extraer el mayor partido posible de la capacidad sonora del **+2A**.

Nota. En los ejemplos que siguen, es importante que usted escriba las expresiones literales exactamente como las mostramos aquí, en mayúsculas y minúsculas; así, por ejemplo, **"ga"** no es lo mismo que **"Ga"**, **"gA"** o **"GA"**.

Escriba esta orden (de momento no se preocupe por lo que significa):

```
PLAY "ga"
```

Suenan dos notas, la segunda ligeramente más alta (aguda) que la primera. La diferencia entre estas notas se llama *tono*. Ahora pruebe:

```
PLAY "g$a"
```

De nuevo han sonado dos notas; la primera era la misma que en el ejemplo anterior, pero el salto hasta la segunda fue menor. Si no ha notado la diferencia, pruebe otra vez el primer ejemplo y luego el segundo.

La diferencia entre las dos notas del segundo ejemplo es un *semitono*.

Cuando se haya familiarizado con los semitonos, pruebe esto:

```
PLAY "gD"
```

Esta diferencia se llama *quinta* y aparece frecuentemente en todos los tipos de música. Con ese ejemplo todavía en sus oídos, escriba:

PLAY "gG"

A pesar de que ahora la diferencia entre las notas ha sido mucho mayor que en el caso de la quinta, de algún modo las notas sonaron bastante más parecidas. Esta diferencia se llama *octava*, y es el punto a partir del cual la música empieza a 'repetirse a sí misma'. No se preocupe demasiado por esto; basta con que recuerde cómo suena una octava.

Hay dos formas de producir música y sonidos con el **+2A**. La más elemental es la orden, un poco espartana, **BEEP**. Su forma es:

BEEP *duración,altura*

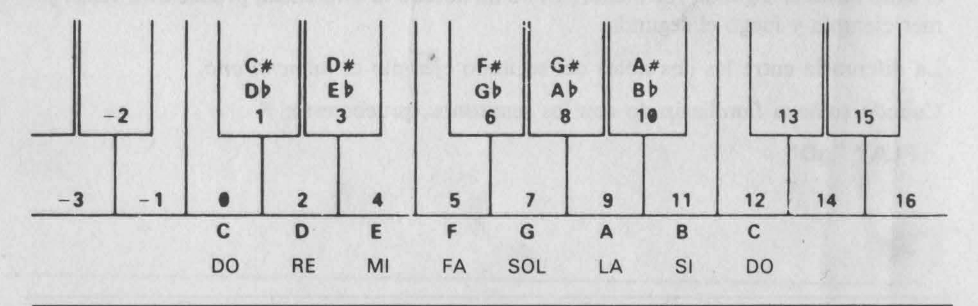
donde, como de costumbre, *duración* y *altura* son expresiones numéricas. La duración se da en segundos; la altura, en número de semitonos por encima de la nota DO medio; los números negativos representan las notas por debajo de la DO medio.

NOTA

Para representar las notas musicales, el **+2A** utiliza la nomenclatura anglosajona. La equivalencia entre ésta y la española es como sigue:

C	DO
D	RE
E	MI
F	FA
G	SOL
A	LA
B	SI

El siguiente diagrama muestra los valores del parámetro *altura* para todas las notas de una octava en el piano.



Por consiguiente, para producir la nota LA que está por encima de la DO medio, con duración de medio segundo, debemos dar la orden:

```
BEEP 0.5,9
```

Para interpretar una escala (por ejemplo, DO mayor) se necesita un programa completo (aunque bastante corto):

```
10 FOR f=1 TO 8
20 READ nota
30 BEEP 0.5,nota
40 NEXT f
50 DATA 0,2,4,5,7,9,11,12
```

Para obtener notas más altas o más bajas que las mostradas en el diagrama, se debe sumar o restar 12 por cada octava que se quiera subir o bajar.

La orden **BEEP** ha sido incluida más que nada por compatibilidad con modelos más antiguos del Spectrum, aunque también puede ser útil para generar efectos sonoros muy cortos o rápidos. Para cualquier programa nuevo que usted desarrolle, la segunda forma de producir sonido, basada en la orden **PLAY**, es, con mucho, la preferible.

PLAY es mucho más flexible que **BEEP**; puede interpretar hasta tres voces en armonía con todo tipo de efectos, y ofrece un sonido de calidad muy superior. Además, es bastante más fácil de usar. Por ejemplo, para interpretar la nota LA posterior a DO medio durante medio segundo, dé la orden:

```
PLAY "a"
```

Para interpretar la escala de DO mayor basta con

```
PLAY "cdefgabC"
```

Observe que la última **C** de este ejemplo es mayúscula. Este hecho indica a la orden **PLAY** que debe interpretar esta nota una octava más arriba que la representada por la **c** minúscula. A propósito, una *escala* es el nombre que se da al conjunto de las notas que hay en una octava. La escala del ejemplo anterior se llama la 'escala de Do mayor' porque es el conjunto de notas entre un DO y el siguiente. ¿Por qué 'mayor'? Hay dos tipos básicos de escala: mayor y menor. Esta terminología es sólo una forma abreviada de describir dos conjuntos diferentes. Por si le interesa, la escala de DO menor suena así:

```
PLAY "cd$efg$a$bC"
```

Cuando una nota va precedida de **$**, suena un semitono más baja (*bemol*); si el prefijo es #, suena un semitono más alta (*sostenido*). La orden **PLAY** cubre 9 octavas; se puede especificar la octava deseada escribiendo la letra **O** mayúscula seguida de un número.

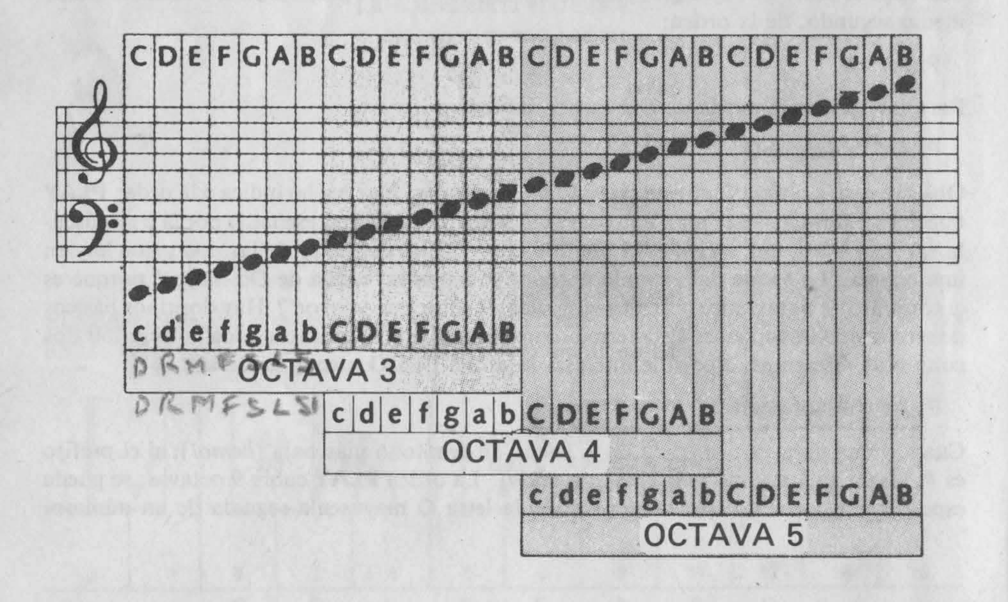
Pruebe este programa:

```
10 LET o$="O5"
20 LET n$="DECcg"
30 LET a$=o$+n$
40 PLAY a$
```

Hay unas cuantas novedades en este programa. En primer lugar, **PLAY** funciona igual de bien con una variable literal que con una constante. En otras palabras, si **a$** ha sido definida con anterioridad, **PLAY a$** produce el mismo efecto que **PLAY "O5DECcg"**. De hecho, el uso de variables en sentencias **PLAY** tiene algunas otras ventajas, y nosotros las emplearemos de aquí en adelante.

Observe también que la cadena **a$** se ha formado por concatenación de dos cadenas más pequeñas: **o$** y **n$**. Aunque esto no representa gran ventaja cuando las cadenas son así de cortas, el hecho es que **PLAY** puede manejar cadenas de una longitud de muchos miles de notas, y la única forma práctica de crear y editar esas cadenas en BASIC es combinar cadenas más pequeñas.

Ahora ejecute el programa anterior. Edite la línea 10 para poner **"O7"** en lugar de **"O5"** y ejecútelo de nuevo. Pruébelo también con **"O2"**. Si no especifica número de octava en una cadena determinada, el **+2A** toma por defecto la octava 5. El siguiente diagrama da las notas y números de octava que corresponden a la 'escala musical bien temperada'.



Hay un solapamiento considerable; por ejemplo, "**O3D**" es lo mismo que "**O4d**". Gracias a esto se puede escribir melodías sin tener que especificar demasiados cambios de octava. Por razones técnicas, algunas notas de las octavas más bajas (0 y 1) no son muy exactas, pero el ordenador trata de desafinar lo menos posible.

PLAY puede producir notas de diferente duración. Edite el programa anterior de forma que la línea 10 sea ahora:

```
10 LET o$="2"
```

y ejecútelo. Después pruebe con otros valores de **o$**, entre "**1**" y "**9**". La longitud de nota se puede especificar en cualquier lugar de la cadena insertando un número del margen de **1** a **9**; la nueva duración queda en vigor hasta que **PLAY** encuentra otro número más adelante. Cada una de estas nueve duraciones de nota tiene un nombre específico en la terminología de la música. La tabla siguiente da los nombres y los símbolos de las notas:

Número	Nombre de la nota	Símbolo musical
1	semicorchea	𝅘𝅥𝅯
2	semicorchea con puntillo	𝅘𝅥𝅯.
3	corchea	𝅘𝅥𝅮
4	corchea con puntillo	𝅘𝅥𝅮.
5	negra	𝅘𝅥
6	negra con puntillo	𝅘𝅥.
7	blanca	𝅗𝅥
8	blanca con puntillo	𝅗𝅥.
9	redonda	𝅝

PLAY también puede producir *tresillos*, que son grupos de tres notas interpretadas en el tiempo de dos. A diferencia de las duraciones de nota sencilla, el número de tresillo sólo afecta a las tres notas siguientes; después de ellas continúa en vigor el anterior número de duración. Los números de tresillo son los siguientes:

Número	Nombre de la nota	Símbolo musical
10	tresillo de semicorcheas	♬
11	tresillo de corcheas	♫
12	tresillo de negras	♩♩♩

Al periodo de tiempo durante el cual no se interpreta ninguna nota se le llama *silencio*. Un silencio se representa por **&**; su longitud es la que esté en vigor para las notas. Edite las líneas 10 y 20 y cámbielas por:

```
10 LET o$="O4"
20 LET n$="DEC&cg"
```

Dos notas ejecutadas juntas forman una *ligadura*. En **PLAY** la ligadura se especifica mediante un signo de subrayado. Por ejemplo, para especificar una DO negra y una DO blanca ligadas se escribe **"5_7c"** (el segundo de los dos números especifica la duración para las notas siguientes).

Hay ocasiones en las que se presenta alguna ambigüedad. Supongamos que una pieza musical necesita la octava 6 y una duración de nota de 2; entonces podríamos pensar que necesitamos:

```
10 LET o$="O62"
```

pero esto no sirve. El ordenador encontrará la **O** y tratará de leer el número siguiente. Como lo que encuentra es el **62**, se detiene y emite el mensaje de error **'n FUERA DE MARGEN'**. Para casos como éste disponemos de una 'nota auxiliar', llamada **N**, que sólo sirve como separador. Así, en la línea 10 deberíamos poner:

```
10 LET o$="O6N2"
```

El volumen es ajustable entre **0** (mínimo) y **15** (máximo) escribiendo el número detrás de la letra **V**. En la práctica, si no se utiliza un amplificador, sólo resultan útiles los números del **10** al **15**, ya que los del **0** al **9** son demasiado suaves. Como ya hemos mencionado, **BEEP** produce un sonido más intenso que un canal de **PLAY**; pero si se hace sonar **PLAY** en los tres canales a volumen **15**, la intensidad será igual a la de **BEEP**.

El manejo de varios canales es muy sencillo; basta con especificar en **PLAY** varias listas de notas, separadas con comas. Pruebe este nuevo programa:

```
10 LET a$="O4cCcCgGgG"
20 LET b$="O6CaCe$bd$bD"
30 PLAY a$,b$
```

En general, no hay diferencia entre los tres canales, y cualquier cadena de notas puede ser reproducida en cualquier canal. La velocidad global de la música, el *tempo*, debe estar en la cadena asignada al canal A (la primera que se especifica tras **PLAY**), pues de lo contrario será ignorada. Para especificar el tempo en número de notas (negras) por minuto, se escribe la letra **T** seguida de un número comprendido entre 60 y 240. El valor estándar es 120, que equivale a dos negras por segundo. Modifique el programa anterior de esta forma:

```
 5 LET t$="T120"
10 LET a$=t$+"O4cCcCgGgG"
20 LET b$="O6CaCe$bd$bD"
30 PLAY a$,b$
```

y ejecútelo varias veces poniendo en la línea 5 tempos diferentes.

Un fenómeno muy frecuente en la música es la repetición de un grupo de notas. Cualquier parte de una cadena puede ser repetida escribiéndola entre paréntesis. Así, si cambiamos la línea 10 por:

```
10 LET a$=t$+"O4(cC)(gG)"
```

el efecto es el mismo que obteníamos antes. Si ponemos el paréntesis de cerrar sin un paréntesis de abrir anterior que le corresponda, se repite indefinidamente todo el tramo previo de la cadena, desde el principio hasta el paréntesis. Esto es aprovechable en los efectos rítmicos y líneas de bajo. Para demostrarlo, pruebe lo siguiente (tendrá que usar BREAK para detener el sonido):

```
PLAY "O4N2cdefgfed)"
```

y después

```
PLAY "O4N2cd(efgf)ed)"
```

Si preparamos una línea de bajo que se repita indefinidamente y después la utilizamos para acompañar una melodía, sería interesante que el acompañamiento terminase al mismo tiempo que la melodía. Afortunadamente, en **PLAY** hay un mecanismo que nos permite lograrlo: si **PLAY** se encuentra la letra **H** en cualquiera de las cadenas que está interpretando, detiene todos los sonidos inmediatamente. Ejecute el siguiente programa (de nuevo tendrá que pulsar BREAK para interrumpirlo):

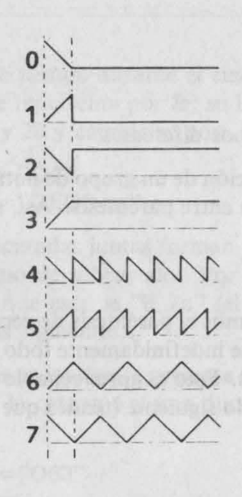
```
10 LET a$="cegbdfaC"
20 LET b$="O4cC)"
30 PLAY a$,b$
```

Ahora modifique la línea 10 de la forma siguiente:

```
10 LET a$="cegbdfaCH"
```

y ejecútelo de nuevo.

Hasta aquí sólo hemos usado notas que comienzan y terminan en el mismo nivel de volumen. El **+2A** puede alterar el volumen de una nota mientras está interpretándola, de forma que, por ejemplo, la nota puede ser intensa al principio y decaer lentamente (como las del piano). Para controlar este efecto se utiliza la letra **W** (de *waveform*, 'forma de onda') seguida de un número entre 0 y 7, junto con una **U** para cada canal al que se quiera aplicar el efecto. Si en un canal determinado se ha especificado volumen con **V**, no responderá a la **U**. El siguiente diagrama muestra la forma en que varía el volumen a lo largo del tiempo para los diversos valores del parámetro de **V**.



0 caída simple y fin
1 ataque simple y fin
2 caída simple y sostenimiento
3 ataque simple y sostenimiento
4 caída repetida
5 ataque repetido
6 ataque-caída repetidos
7 caída-ataque repetidos

El siguiente programa interpreta la misma nota con cada uno de estos valores. Trate de «escuchar» las formas ilustradas en el diagrama.

```
10 LET a$="UX1000W0C&W1C&W2C&W3C&W4C&W5C&W6C&W7C"
20 PLAY a$
```

La **U** activa los efectos y la **W** selecciona la forma de onda. La cláusula **X1000** establece cuánto tiempo deben durar los efectos (su parámetro tiene que estar entre 0 y 65535). Si no se incluye la **X**, el **+2A** escogerá el valor más largo. Las formas que llegan a estabilizarse (1 a 3 en la tabla anterior) funcionan mejor con valores de **X** en torno a 1000, mientras que las periódicas (4 a 7) son más eficaces con valores pequeños, tales como 300. Pruebe diversos valores de **X** en el programa anterior para hacerse una idea de cómo funciona cada uno.

La orden PLAY no está limitada a las notas puramente musicales. Hay también tres generadores de 'ruido blanco' (ruido blanco es, más o menos, el ruido que produce la radio en FM o el televisor cuando no están sintonizados). Cualquiera de los tres canales puede interpretar notas, ruido blanco o una mezcla de ambas cosas. Para seleccionar esta mezcla se especifica la letra **M** seguida de un número (del 1 al 63). El significado del número es el que se indica en la siguiente tabla.

	Canales de tono			**Canales de ruido**		
	A	B	C	A	B	C
Número	1	2	4	8	16	32

Para obtener el número que se debe especificar tras **M**, se toma nota de los números que corresponden a los efectos que se desea activar y luego se los suma. Por ejemplo, si quisiéramos ruido en al canal A, tono en el B y ambas cosas en el C, tendríamos que sumar 8, 2, 4 y 32 para obtener 46 (el orden de los canales es el orden en que ponemos las cadenas tras la orden **PLAY**). Los mejores efectos se obtienen en el canal A; no se prive de experimentar.

A estas alturas ya estará usted escribiendo sinfonías. Sin embargo, cuando las cosas se complican, puede ser difícil recordar qué tramo concreto de una cadena es responsable de cierta parte de la música. Para aliviar este problema, la cadena puede incluir comentarios escritos entre signos de admiración !. Por ejemplo,

```
1098 LET z$=z$+"CDcE3Ge4_6f! fin del compás 75 !egeA"
```

La orden **PLAY** sencillamente se salta los comentarios y los ignora.

Si usted dispone de un instrumento musical electrónico con MIDI, el **+2A** puede controlarlo con la orden **PLAY**. Hasta ocho canales de música pueden ser enviados a los sintetizadores, tambores y secuenciadores. La orden **PLAY** se construye exactamente como hemos explicado en esta sección, con la única diferencia de que cada cadena debe incluir una **Y** seguida de un número (entre 1 y 16). Este número controla a qué canal son asignados los datos de la música. Se puede usar hasta ocho cadenas; las tres primeras seguirán siendo interpretadas a través del televisor, como antes, por lo que puede ser conveniente bajar el volumen del aparato. También se puede enviar códigos de programación del MIDI a través de la orden **PLAY**, usando para ello una **Z** seguida del número del código. Las velocidades (intensidad sonora) se calculan y envían como 8 veces el valor de **V** (así, **V6** enviará el número 48 como velocidad).

Por ejemplo, para enviar una pequeña melodía a un sintetizador de cuatro voces (después de consultar al manual del sintetizador para averiguar cómo se asigna los canales del MIDI

a las diferentes voces), podemos usar una orden **PLAY** con cuatro cadenas, cada una de las cuales empezará por la letra **Y** y un número. El siguiente programa ilustra la orden **PLAY** en todo su esplendor:

```
10 LET a$="Y1T100O2(((1CCg$b))
   (($E$E$b$D))((FFC$E))((GGDF
   ))))"
20 LET b$="Y2O5N&&&&C$bfG)"
30 LET c$="Y3O4((3C&)C&1CCDD(3
   $E&)$E&1$E$EEE(3F&)F&1FF$G$
   G(3G&)G&1GG$EC))
40 LET d$="Y4N9&&&&&&&&&(9EGF7b
   5CD))"
50 PLAY a$,b$,c$,d$
```

Tabla de resumen

Para concluir esta sección, vamos a dar una lista de los parámetros que pueden ser incluidos en las cadenas de **PLAY**, junto con los valores que pueden tomar:

Cadena	Función
a...g **A...G**	Especifican el tono de la nota, dentro de la octava actual
$	Especifica que la nota siguiente debe ser convertida en bemol
#	Especifica que la nota siguiente debe ser convertida en sostenido
O*n*	Especifica el número de octava (*n* entre 0 y 8)
1...12	Especifica la duración de las notas
&	Especifica un silencio
_	Especifica una ligadura
N	Separador entre números
V*n*	Especifica el volumen de las notas (*n* entre 0 y 15)
W*n*	Especifica el efecto de variación de volumen (*n* entre 0 y 7)
U	Introduce el efecto de variación de volumen en una cadena
X*n*	Especifica la duración del efecto de variación de volumen (*n* entre 0 y 65535)
T*n*	Especifica el tempo de la música (*n* entre 60 y 240)
(...)	Los paréntesis especifican que la frase escrita entre ellos debe ser repetida
!...!	Los signos de admiración especifican que el comentario escrito entre ellos debe ser ignorado

Cadena	Función
H	Interrumpe la generación de sonido
M*n*	Especifica qué canal o canales, musicales o de ruido, debe usar **PLAY** (*n* entre 1 y 63)
Y*n*	Especifica qué canal del MIDI se debe usar (*n* entre 1 y 16)
Z*n*	Especifica un código de programación del MIDI (*n* es el número del código)

Sección 20
Operaciones con los ficheros

Temas tratados:

- Discos y unidades para el **+2A**
- Unidades de disquete mecánicas y disco de RAM
- **FORMAT**
- Nombres de fichero
- **SAVE, LOAD**
- Catálogo del disco: **CAT**
- Caracteres polivalentes
- **MERGE**
- **LINE, DATA, CODE, SCREEN$**
- Borrado y cambio de nombre de los ficheros
- **ERASE, MOVE**
- Atributos de los ficheros
- Copia de ficheros: **COPY**
- **SCREEN$, LPRINT, SPECTRUM FORMAT**
- El disco de RAM
- Operaciones con las cintas
- **SAVE, LOAD, MERGE**
- Catálogo de la cinta: **CAT**

En el Capítulo 6 hemos visto las órdenes de gestión de ficheros más sencillas (v.g., **SAVE**, **LOAD**, etc.). En esta sección vamos a estudiar el mismo tema en mayor profundidad.

Gran parte de la información que daremos sólo es aplicable a sistemas en los que se haya instalado una o dos unidades de disquete externas. No obstante, lo relativo al disco de RAM (unidad M) es válido para todos los equipos. Al final de la sección explicaremos detalladamente la forma de seleccionar la unidad por defecto. También trataremos el sistema de almacenamiento en cinta, de modo que todos los usuarios deberían leer esos apartados, en particular si no disponen de unidad de disquete.

Discos y unidades para el +2A

Si desea conectar una unidad de disquete externa al **+2A**, debe adquirir el modelo AMSTRAD FD-1, junto con un interfaz adecuado (AMSTRAD SI-1). Consulte a este respecto el Capítulo 10, 'Periféricos para el **+2A**'.

La unidad AMSTRAD FD-1 sólo admite 'discos compactos de 3 pulgadas'. Para garantizar la integridad de los datos y los programas, recomendamos el uso de discos CF-2 de AMSOFT; no obstante, los discos de los principales fabricantes son igualmente adecuados.

Unidades de disquete mecánicas y disco de RAM

Las unidades de disco son dispositivos que podemos usar para grabar y cargar nuestros propios programas y para cargar los programas de distribución comercial (juegos, utilidades, etc.). Dado que es posible conectar **dos** unidades de disco externas, BASIC necesita un medio de distinguir una de la otra. La primera unidad es la **A**: (el signo de dos puntos sirve para que +3 BASIC entienda que cuando escribimos '**A:**' queremos decir 'unidad de disco A'). La segunda unidad es la **B**:.

Puesto que el procesador del **+2A** sólo puede acceder a 64K de la memoria en un momento dado, el resto de la memoria RAM del **+2A** (otros 64K) es utilizado como si constituyera otra unidad de disco. Es el denominado *disco de RAM*, identificado por **M**:. Todas las órdenes (excepto **FORMAT**) que funcionan con las dos unidades mecánicas (A y B) pueden ser aplicadas también a la unidad M. El disco de RAM es mucho más rápido que las unidades mecánicas. Su principal desventaja consiste en que, al igual que la memoria dedicada a los programas, el contenido del disco de RAM se borra en cuanto pulsamos el botón **REINIC** o apagamos el ordenador. (Sin embargo, la orden **NEW** de BASIC no afecta al disco de RAM.)

Aunque sólo tengamos conectada una unidad de disco externa, podemos utilizar el ordenador como si también estuviese presente la segunda. En efecto, si pedimos al **+2A** que realice cualquier operación con la unidad B, aparecerá en la pantalla el mensaje:

Introduzca en la unidad el disco
para B y luego pulse una tecla

Lo que debemos hacer entonces es introducir en la unidad A el disco que habríamos puesto en la unidad B si ésta existiese, y luego pulsar una tecla (por ejemplo, **INTRO**). A partir de ese momento, el **+2A** trata la unidad A como si verdaderamente fuera la unidad B. Si más tarde necesita realizar alguna operación con el disco que originalmente estaba en la unidad A, nos pedirá el cambio de disco mediante el siguiente mensaje:

Introduzca en la unidad el disco
para A y luego pulse una tecla

Esta técnica será particularmente útil cuando nos sirvamos de la orden **COPY** (descrita más adelante).

Ahora que ya sabemos de qué unidades disponemos y cuáles son sus nombres, vamos a estudiar las formas de utilizarlas. Para empezar, transcriba el programa que vimos al final de la Sección 16 (el que llenaba la pantalla con cuadrados de colores):

```
10 POKE 22527+ RND *704, RND *127
20 GOTO 10
```

Éste es el programa que vamos a grabar en disco.

Si no tiene conectada ninguna unidad externa, no es necesario que lea lo que vamos a decir a continuación acerca de la orden **FORMAT**, ya que el disco de RAM no necesita ser formateado. (De hecho, no se puede aplicar la orden **FORMAT** a la unidad M.)

Tal como explicábamos en el Capítulo 6, antes de poder grabar programas en un disco nuevo tenemos que formatearlo, o sea, someterlo a un proceso de preparación que se realiza con la orden **FORMAT**. Esta orden borra absolutamente todo lo que pueda haber en el disco y lo deja preparado para +3 BASIC. Naturalmente, hay que poner sumo cuidado en no formatear por error un disco que contenga programas que queramos conservar.

Nota. En el **+2A** las dos caras de cada disco son independientes. Por eso, cuando decimos 'disco' generalmente nos referimos a una de sus caras (la que quede arriba al introducir el disco en la unidad). Siempre que hablemos de 'introducir un disco' se debe entender que el disco debe ser introducido en la unidad con la cara que queramos utilizar (formatear, copiar, grabar, etc.) hacia arriba.

Para formatear el disco dé la orden:

```
FORMAT "a:"
```

Si al pulsar [INTRO] todavía no ha introducido el disco en la unidad, el **+2A** emitirá el mensaje '**UNIDAD NO PREPARADA**'. En tal caso, introduzca el disco (con la cara que desee formatear hacia arriba) y repita la orden.

Dicho sea de paso, muchas órdenes de manejo de discos pueden producir el mensaje '**UNIDAD A: NO PREPARADA**' (en ocasiones seguido de '**– ¿REINTENTAR, IGNORAR O CANCELAR**'). Esto siempre significa que no hemos introducido el disco en la unidad.

Cuando un mensaje va seguido de '**– ¿REINTENTAR, IGNORAR O CANCELAR**', las opciones de que disponemos son las tres siguientes:

- La primera es poner remedio al problema y reintentar la operación. Por ejemplo, si el mensaje es '**UNIDAD A: NO PREPARADA**', tendremos que introducir el disco y luego pulsar [R].
- Si estamos en medio del proceso de copia de un fichero largo y aparece, por ejemplo, el mensaje '**FALTA MARCA DE DIRECCIONES**', esto normalmente quiere decir que

el disco que está siendo leído ha sido dañado. Lo aconsejable en este caso sería pulsar unas cuantas veces [R]. Si el error persiste, no hay duda de que el disco está estropeado. Lo que interesa entonces es poner a salvo los datos no deteriorados, así que debemos pulsar [I] (para ignorar el error). Esto no garantiza que el sistema de disco pueda leer los datos intactos; no es más que un último intento de salvar lo que se pueda.

- Finalmente, cuando se produce un error, muchas veces nos damos cuenta de que no tiene sentido seguir adelante con la operación emprendida. En tal caso debemos pulsar [C] para pedirle al **+2A** que abandone la orden que le habíamos dado. Generalmente BASIC emitirá un nuevo mensaje de error, con un texto muy parecido al mensaje original.

Pero volvamos a nuestro intento de formatear el disco. Si hemos introducido en la unidad (generalmente por error) un disco que ya está formateado, el **+2A** lo detecta y nos avisa con el mensaje:

Ya formateado. Tecla A para
abandonar/otra para continuar

Ésta es una medida de protección con la que el ordenador nos da la oportunidad de abandonar el proceso antes de borrar un disco. Para ello basta con pulsar [A]. Sin embargo, si efectivamente queremos formatear el disco, podemos pulsar cualquier otra tecla (por ejemplo, [INTRO]).

Al cabo de unos 30 segundos aparece el mensaje '**0 OK, 0:1**' habitual. El disco ha quedado preparado y normalmente no tendremos que volver a formatearlo. No obstante, podemos hacerlo si alguna vez queremos borrarlo completamente, pero siempre teniendo en cuenta que éste es un proceso irreversible.

Los discos no son, ni mucho menos, invulnerables. Un disco se puede estropear por el hecho de que una partícula de polvo se adhiera a su superficie sensible, o porque haya estado sometido a la influencia de un campo magnético (por ejemplo, el producido por un televisor o un teléfono), o porque lo hayamos extraído de la unidad cuando el ordenador estaba leyendo o escribiendo en él. Cuando un disco se estropea sus datos quedan *corruptos*. Un disco corrupto producirá errores en las operaciones de lectura y escritura; antes de intentar grabar en él debemos reformatearlo.

Se puede tratar de recuperar los ficheros de un disco corrupto copiándolos uno a uno (con **COPY**) hacia otro disco. Si no conseguimos copiar algún fichero, podemos darlo por perdido para siempre. Es conveniente guardar un par de copias de todos los ficheros importantes (en un disco diferente del original): una para el trabajo diario, otra como reserva para cuando ocurra lo «imposible». La práctica de copiar periódicamente los programas y los datos es lo que se llama 'hacer copias de seguridad'. Es un hábito que conviene adquirir, pues a la larga nos va a ahorrar muchos disgustos. (Puede interesar hacer las copias de seguridad en cinta, ya que éste es un soporte más barato.)

A diferencia de lo que ocurre en otros ordenadores, **FORMAT** es una orden de +3 BASIC y puede ser utilizada como cualquier otra. No afecta al *programa actual* (es decir, al que tenemos en la memoria del ordenador en el momento de ejecutarla). Por consiguiente, estamos preparados para grabar el programa que transcribimos al principio de esta sección.

Si no tiene conectada ninguna unidad externa, seleccione el disco de RAM con la orden:

SAVE "m:"

Cualquiera que sea la configuración de su equipo, dé la siguiente orden:

SAVE "cuadrado"

La palabra '**cuadrado**' es sencillamente el nombre que utilizamos para 'etiquetar' el programa en el disco. Para evitar confusión, el sistema de almacenamiento en disco exige que demos un nombre a todos los programas. Estos nombres son algo diferentes de los que podemos usar para almacenar programas en cinta.

Nombres de fichero

El formato de los nombres de fichero que podemos usar en el **+2A** es el mismo que en el sistema operativo CP/M de Digital Research. Gracias a esta compatibilidad, podemos usar en otros ordenadores los discos creados en el **+2A**. Esto proporciona una forma de transferir datos entre el **+2A** y el sistema operativo CP/M, que seguramente será aprovechada por los usuarios que escriban programas en código de máquina o que deseen procesar con un programa de CP/M los ficheros creados con un procesador de texto en el **+2A**. (Es muy improbable que los programas de +3 BASIC puedan ser convertidos de un ordenador a otro por este método.)

Los nombres de fichero pueden ser tan sencillos como el que usábamos antes (**CUADRADO**), o incluso más sencillos: por ejemplo, **C**. Sin embargo, un nombre de fichero de CP/M puede constar de hasta cuatro partes: el *número de usuario*, la *letra de la unidad*, el *nombre* y el *tipo*. Cada una de estas partes es un *campo*.

Si usted no sabe qué es el número de usuario, quizá sea preferible que no intente aprenderlo ahora, pues no va a necesitar este concepto para trabajar con el **+2A**. En las máquinas de CP/M con discos de gran capacidad (por ejemplo, discos duros), a veces conectadas a varios terminales, los números de usuario permiten distribuir los ficheros en varias secciones (áreas de usuario), de modo que no haya necesidad de manejar un enorme directorio, quizá con miles de ficheros. Sin embargo, en el **+2A** los discos no pueden contener más de 64 ficheros, de modo que la utilización de números de usuario no tiene demasiado interés. No obstante, es posible especificar las áreas de usuario en todas las órdenes de gestión de discos del **+2A**. Los nombres de fichero tienen entonces la siguiente forma:

número-de-usuario letra-de-unidad:nombre-de-fichero

donde *número-de-usuario* es un número del margen de 0 a 15 y la *letra-de-unidad* es **A**, **B** o **M**. Si se incluye el número de usuario, es obligatorio especificar también la letra de

unidad (que no puede ser la A ni la B en los equipos que no dispongan de unidad externa). Por ejemplo, para grabar el programa actual en el área de usuario número 5 de la unidad A daríamos la orden:

SAVE "5a:cuadrado"

Cuando grabamos en cinta, el número de usuario y la letra de la unidad serán consideradas como parte del nombre del fichero (y cuentan a efectos del límite de longitud del nombre, que no puede constar de más de 10 caracteres). En la cinta no tiene sentido el concepto de área de usuario.

Uno de los problemas que acarrea la utilización de áreas de usuario es que fácilmente olvidamos en qué área hemos grabado un fichero determinado, y encontrarlo puede llevar bastante tiempo (ya que la orden **CAT** sólo da la lista de los ficheros de un área cada vez que la ejecutamos).

Como acabamos de decir, la *letra-de-unidad* puede ser **A**, **B** o **M**. Observe que la letra tiene que ir seguida del signo de dos puntos (v.g., **a:cuadrado**). Si no especificamos este campo, el **+2A** supone que queremos usar la denominada 'unidad *implícita*', 'unidad *activa*' o 'unidad *por defecto*'. (Al encender el ordenador, +3 BASIC adopta la unidad A como unidad implícita, suponiendo que esté presente.) Por consiguiente, si suponemos que efectivamente está conectada la unidad, la orden

SAVE "cuadrado"

es equivalente a

SAVE "a:cuadrado"

siempre que no hayamos cambiado de unidad implícita.

En ausencia de la unidad de disco externa, la unidad implícita es el magnetófono de cassette. En tal caso, las dos órdenes anteriores graban el programa en la cinta (con dos nombres diferentes), emitiendo previamente el mensaje '**PREPARE LA CINTA Y PULSE INTRO**'. No se preocupe si no había puesto cinta en el magnetófono y el programa no ha sido grabado. En los siguientes ejemplos utilizaremos el disco de RAM (unidad M).

Las órdenes **SAVE** y **LOAD** tienen sendas formas especiales con las que podemos cambiar de unidad implícita sin producir ningún otro efecto. Si **SAVE** o **LOAD** van seguidas de una especificación de fichero que sólo consiste en una letra de unidad y el signo de dos puntos, la unidad especificada se convierte en nueva unidad implícita. Por ejemplo,

SAVE "m:"
SAVE "cuadrado"

selecciona la unidad M como nueva unidad implícita y graba el programa en ella. (En cambio, la orden **SAVE "m:cuadrado"** grabaría el programa en la unidad M, pero la unidad implícita seguiría siendo la misma.)

Para volver a seleccionar la unidad A daremos la orden:

SAVE "a:"

(Si no tiene conectada la unidad externa, ignore esta orden y mantenga la M como unidad implícita.)

Debe quedar claro que las órdenes **SAVE** y **LOAD** seguidas de solamente una letra de unidad (y el signo de dos puntos) no hacen más que establecer la unidad implícita. En particular, no graban ni cargan ningún fichero; para que lo hagan es necesario especificar un nombre de fichero.

Cuando utilizamos discos (incluido el de RAM), el campo del *nombre* es imprescindible en la especificación de un fichero. Puede contener entre 1 y 8 caracteres, elegidos de entre los siguientes:

letras:	**abcdefghijklmnopqrstuvwxyz** (mayúsculas o minúsculas)
dígitos:	**0123456789**
otros signos:	**"#$'@↑_{ }~'**

Las letras mayúsculas y minúsculas son equivalentes en los nombres de fichero, de modo que el nombre **EJEMPLO** es idéntico a **ejemplo**, a **Ejemplo**, etc.

El último campo, el *tipo*, es opcional y puede contener hasta 3 caracteres. Se lo suele utilizar para agrupar con un mismo *tipo* todos los ficheros que contienen información de la misma clase y distinguirlos así de los demás. Si se incluye este campo, tiene que ir precedido por un punto. (Algunas versiones de BASIC asignan automáticamente un tipo cuando el usuario no especifica ninguno; +3 BASIC no lo hace.) Es conveniente utilizar tipos bien escogidos; frecuentemente se asigna **.BAS** a los programas de BASIC y **.BIN** a los ficheros binarios. Así, el programa de nuestro ejemplo lo podríamos haber grabado con la orden:

```
SAVE "cuadrado.bas"
```

Los caracteres * y ? tienen un significado especial y no pueden ser incluidos en los nombres de fichero que especificamos con las órdenes **SAVE** y **LOAD**. En cambio, hay otras órdenes de gestión de ficheros que sí los admiten (con ese significado especial), como veremos más adelante.

Por ejemplo, las siguientes combinaciones de caracteres son válidas como nombres de fichero:

```
z
cuadrado
m:imagen.bin
a:felipe
13a:hola
0M:MAYUSC
prueba.bas
puntos
cristal.xyz
a:a.a
```

(Si no tiene conectada la unidad externa, algunos de estos nombres producirán el mensaje de error **'UNIDAD NO ENCONTRADA'**.)

No son válidos, en cambio, los siguientes nombres:

ahora no (porque contiene un espacio)
(prueba) (por los paréntesis)
/<>−+=!& (porque no puede contener ninguno de estos caracteres)
demasiados (porque contiene más de 8 caracteres)
pero... (porque no debería contener más de un punto)
7:dudoso (porque si se especifica número de usuario hay que incluir también la letra de unidad)

Catálogo del disco

Quizá se haya dado usted cuenta de que hemos grabado el mismo programa dos veces con dos nombres diferentes (**CUADRADO** y **CUADRADO.BAS**). Seguramente habrá pensado que sería interesante poder averiguar qué programas tenemos almacenados en un disco determinado. Para eso disponemos de la orden **CAT**. Esta orden genera una lista (catálogo) de los ficheros que hay en el disco.

Pulse INTRO y escriba:

CAT

El **+2A** echa un rápido vistazo al disco (suponiendo que la unidad implícita sea la A y esté conectada) y escribe en la pantalla la lista de los ficheros que encuentra en él. Si la unidad implícita es la M, **CAT** da la lista de los ficheros del disco de RAM. La lista está en orden alfabético e indica, junto al nombre, el tamaño de cada fichero (en número entero de kilobytes, redondeando hacia arriba). Al final da también el espacio que queda libre en el disco.

Tal como la hemos usado, la orden **CAT** ha dado la lista de los ficheros de la unidad implícita. Si queremos el catálogo de otra unidad debemos especificar su letra. Por ejemplo,

CAT "b:"

(Si no está conectada la unidad externa, esta orden producirá el mensaje de error **'UNIDAD NO ENCONTRADA'**.)

Por lo tanto, para pedir el catálogo de una unidad cualquiera sin seleccionarla como unidad implícita, tenemos que especificar su letra en la orden **CAT**. Con las órdenes:

LOAD "b:"
CAT

también obtendríamos el catálogo de la unidad B, pero habríamos dejado seleccionada esta unidad como unidad implícita. (Si ha dado estas órdenes, vuelva a seleccionar la unidad M, con **LOAD "m:"**, antes de continuar.)

Para probar las diversas formas de la orden **CAT** necesitaremos unos cuantos ficheros en el disco. Grabe varias veces el mismo programa con las órdenes de la siguiente lista:

```
SAVE "cuadrado"
SAVE "cuadrado.bas"
SAVE "bingo"
SAVE "sal"
SAVE "carta.bin"
SAVE "tren.bak"
SAVE "sumas.ebs"
SAVE "caramelo.bas"
```

No importa que el disco de RAM quede lleno de copias de un mismo programa. Enseguida veremos cómo borrar ficheros.

Caracteres polivalentes

Si un disco contiene muchos ficheros, generalmente será preferible obtener listados parciales que incluyan solamente los ficheros de interés. Por ejemplo, si sólo nos interesan los ficheros cuyo tipo es **BAS**, podemos dar la orden:

```
CAT "*.BAS"
```

El asterisco (*) que hemos usado es un *carácter polivalente*. Cuando en la orden **CAT** especificamos un nombre de fichero o una *plantilla* (construida con caracteres polivalentes), la lista que obtenemos contiene solamente los ficheros que responden a la especificación. Si incluimos el carácter polivalente * (en el campo del nombre o en el del tipo), el ordenador lo interpreta como 'cualesquiera caracteres que haya entre esta posición y el final del campo'. Así, en la orden anterior, **CAT** ha buscado todos los ficheros de tipo **BAS**, sin importarle qué caracteres tuvieran en el campo del nombre.

Si no hay ningún fichero que encaje en la especificación, la orden **CAT** emite el mensaje '**NINGUN FICHERO ENCONTRADO**' e indica el espacio que queda libre en el disco. Si especificamos la plantilla *.* (o sea, **CAT "*.*"**), o no especificamos nada (o sea, **CAT**), y la orden responde con ese mensaje, es porque el disco está vacío. Si se trata de la unidad A o la B, el espacio libre será 173K (a menos que el disco proceda de otro ordenador). Si la unidad es M, el espacio libre normalmente será 62K (o 58K si tenemos conectada alguna unidad externa). Si pedimos el catálogo de un disco de distribución comercial (por ejemplo, de un juego), la lista puede no mostrar ningún nombre de fichero y sí un espacio libre bastante reducido; esto indica que el fabricante del programa ha tomado ciertas medidas de protección contra copias fraudulentas.

Con la orden:

```
CAT "c*.bas"
```

obtenemos la lista de todos los ficheros cuyo nombre empieza por **C** y cuyo tipo es **BAS**. En nuestro disco el listado incluirá **CUADRADO.BAS** y **CARAMELO.BAS**, porque solamente estos nombres de fichero encajan en la plantilla **c*.bas**.

Los caracteres polivalentes también pueden figurar en el campo del tipo (pero no en el del número de usuario ni en el de letra de unidad). Si queremos la lista de todos los ficheros cuyo nombre sea **CUADRADO**, sin importarnos el tipo, podemos dar la orden:

CAT "cuadrado.*"

Si queremos la lista de los ficheros cuyo nombre empieza por **C** y cuyo tipo empieza por **B** usaremos la orden:

CAT "c*.b*"

La plantilla *.* significa 'todos los ficheros', de modo que **CAT "*.*"** es equivalente a **CAT** a secas.

De estos ejemplos se puede deducir que el carácter * tiene que ser el último en el campo en que intervenga, y que, como decíamos antes, significa 'los caracteres que haya entre esta posición y el final del campo'.

A veces necesitamos plantillas un poco más restrictivas. El otro carácter polivalente es el signo de interrogación, **?**. Significa 'cualquier carácter que haya en esta posición' y puede intervenir en el campo del nombre y en el del tipo.

De este modo, la orden

CAT "?al"

da la lista de los ficheros que carecen de tipo y cuyo nombre tiene una longitud de tres caracteres y termina en **al**. En nuestro disco el único fichero que encaja en esta plantilla es **SAL**. Si existieran, también encajarían los ficheros **TAL**, **MAL**, etc., pero no **SAL.BAS** ni **SALE**. A diferencia de lo que ocurría con *, el carácter **?** puede aparecer varias veces en un mismo campo. Por ejemplo,

?.bas	(nombre de una sola letra, tipo **BAS**)
c?adrado.*	(el segundo carácter del nombre puede ser cualquiera; los demás tienen que ser precisamente los especificados; el tipo es indiferente)
tr??.?a?	(nombre de cuatro caracteres que empiece por **TR**; tipo de tres caracteres, el segundo de los cuales tiene que ser **A**)
????????.???	(equivale a *.*)

Si se ha conectado una impresora, puede ser útil enviarle el catálogo del disco para tener una copia en papel (sobre todo si el disco contiene muchos ficheros). Para ello tenemos que desviar la salida generada por **CAT** hacia el canal número 3 (v. Sección 22). La orden tiene esta forma:

CAT #3

Si se desea sólo un listado parcial, se puede dar una orden del siguiente estilo:

CAT #3,"m:*.bas"

(Estas dos últimas órdenes sólo pueden funcionar si la impresora está conectada y 'en línea'. Para abandonarlas se pulsa **BREAK**.)

Cualquier forma de la orden **CAT** puede terminar en **EXP** (por ejemplo, **CAT "m:" EXP**). '**EXP**' es abreviatura de 'expandido' y produce un listado ampliado en el que se incluye información sobre los *atributos* de los ficheros. En esta versión, **CAT** indica si los ficheros son 'de sistema' o 'de archivo', o si están protegidos contra escritura; véase el apartado 'Atributos de los ficheros' más adelante.

(Hay otra versión de la orden **CAT** que explicaremos en el apartado 'Catálogo de la cinta'.)

Hay una diferencia muy importante entre reinicializar el ordenador y ejecutar una orden **NEW**. Al pulsar el botón **REINIC** se borra toda la memoria RAM, incluidos los ficheros que hayamos almacenado en la unidad M. En cambio, **NEW** no afecta en absoluto a los ficheros de la unidad M. Para comprobarlo, dé la orden:

NEW

seleccione la opción **+3 BASIC** en el menú de presentación y escriba:

LOAD "m:cuadrado"

La orden **LOAD** lee en el disco un programa (junto con sus variables), borrando previamente de la memoria el programa actual (y sus variables). (Sin embargo, si el fichero especificado en **LOAD** no se encuentra en el disco, la orden no borra el programa actual.) Al igual que **SAVE**, **LOAD** exige que especifiquemos un nombre de fichero no vacío. Si usted ya ha trabajado con la cinta, posiblemente estará acostumbrado a la orden **LOAD ""**, que significa 'cargar el próximo programa que se encuentre en la cinta'. Sin embargo, en el disco no tiene sentido el concepto de 'próximo programa'; por eso, si no especificamos un nombre de fichero, el ordenador no sabe qué programa tiene que cargar y emite un mensaje de error. Si no recuerda qué nombre dio al programa en el momento de grabarlo, use la orden **CAT** para averiguar qué ficheros hay en el disco. (Es una buena costumbre elegir los nombres de los programas de forma que den una idea de cuál es su contenido o misión; por ejemplo, si vemos en el disco el nombre **TENIS.BAS**, seguramente recordaremos que es un programa de juego, mientras que si le hubiéramos llamado solamente **T** el nombre no nos serviría de gran ayuda.)

Cuando se tiene conectada la unidad externa, hay una forma más directa de cargar programas (por ejemplo, juegos): usar la opción **Cargador** del menú de presentación. Sin embargo, este método no vale para todos los programas. En efecto, al seleccionar esa opción, el ordenador busca en el disco de la unidad A un programa llamado *. Si existe, lo carga y lo pone en marcha. El programa tiene que estar escrito en código de máquina y haber sido grabado de una forma especial (BASIC no puede grabar programas con ese nombre,

ya que **SAVE** no admite el carácter * como nombre de fichero). Así pues, este método está reservado para los programas de distribución comercial y para los usuarios que sepan programar en código de máquina.

Si el ordenador no encuentra el fichero *, busca en la unidad A un fichero que se llame **DISK**. (Éste sí puede ser un programa de BASIC.) Si encuentra ese fichero, lo carga y queda a la espera de nuevas órdenes. En esta situación, si pulsamos [INTRO], el **+2A** simplemente vuelve a cargar el mismo programa.

Para ejecutar o editar el programa cargado de esta manera, se pulsa [↓] y luego [INTRO] a fin de seleccionar la opción **+3 BASIC** del menú de presentación.

Si tampoco existe el fichero **DISK** (o si no hay ningún disco en la unidad A, o si ésta no está conectada), el ordenador supone que queremos cargar un programa de cinta y emite el mensaje:

Introduzca la cinta y pulse PLAY
Cancelar: pulse BREAK dos veces

Ésta es la forma que recomendamos para cargar programas de cinta escritos para el Spectrum **+2A** (así como los del Spectrum **+3**, Spectrum **+2** o del Spectrum 128); v. Capítulo 4.

Ya hemos mencionado que **LOAD** borra el programa actual y sus variables cuando el **+2A** consigue cargar un nuevo programa. Sin embargo, existe otra orden, **MERGE**, que sólo borra las líneas (o las variables) del programa antiguo cuando su número (o nombre) existe también en el nuevo. Borre el programa actual con la orden **NEW** y copie el de lanzamiento de dados que vimos en la Sección 11 de este capítulo; luego grábelo con las órdenes:

```
LOAD "m:"
SAVE "m:"
SAVE "dados"
```

Dé otra vez la orden **NEW** y escriba y ejecute este programa:

```
1 PRINT 1
2 PRINT 2
10 PRINT 10
20 LET x=20
```

Ahora dé la siguiente orden:

```
MERGE "dados"
```

Si lista el programa, comprobará que las líneas 1 y 2 han sobrevivido y que las líneas 10 y 20 han sido reemplazadas por las de **dados**. Observe que también ha sobrevivido la variable **x** (como puede comprobar con **PRINT x**).

A estas alturas ya hemos visto y probado las formas más sencillas de cinco de las órdenes relacionadas con la gestión de las unidades de disco:

FORMAT Prepara los disquetes nuevos de forma que el ordenador pueda grabar en ellos. También sirve para borrar completamente los discos usados. (Sólo funciona si se tiene conectada alguna unidad de disquete externa.)

SAVE Almacena el programa y sus variables en el disco.

LOAD Borra el programa y las variables actuales y carga el programa y las variables leídos en el disco.

MERGE Es similar a **LOAD**, con la diferencia de que sólo borra las líneas y variables del programa antiguo que también existen en el nuevó.

CAT Da una lista de los ficheros que encuentra en el disco.

Hay una variante de **SAVE** que tiene la forma:

SAVE *nombre-de-fichero* **LINE** *número-de-línea*

Un programa grabado con esta orden queda almacenado de tal manera que, al cargarlo, +3 BASIC salta al *número-de-línea* e inicia la ejecución automáticamente.

Si no tiene conectada la unidad externa, **no** pruebe el siguiente ejemplo.

Dé la orden **NEW** para borrar de la memoria el programa actual y luego copie el siguiente:

```
10 PRINT "programa funcionando"
20 PLAY "cdefgabC"
```

Ahora grabe este programa con la orden:

```
SAVE "disk" LINE 10
```

Ejecute otra vez la orden **NEW** para borrar el programa de la memoria. Cuando aparezca el menú de presentación, pulse [INTRO]. De esta forma habrá seleccionado la opción **Cargador** del menú de presentación. El ordenador se pone a buscar en el disco el programa **DISK**; cuando encuentra el pequeño programa que acabamos de grabar con ese nombre, lo carga y, puesto que lo habíamos grabado con la opción **LINE 10**, inicia su ejecución (a partir de la línea 10). Si ahora pulsamos [INTRO] el **+2A** vuelve a cargar y a ejecutar el mismo programa.

Si usted quiere editar el programa cuando ya ha sido ejecutado, pulse [↓] e [INTRO] (para seleccionar la opción **+3 BASIC** en el menú de presentación).

Observe que si el programa **DISK** cargado por este procedimiento (o sea, con la opción **Cargador**) no es de ejecución automática, para ponerlo en marcha o editarlo tendremos que seleccionar la opción **+3 BASIC** cuando el ordenador haya terminado de cargarlo.

Hasta ahora lo único que hemos grabado ha sido programas (con sus variables). También podemos grabar información de otros dos tipos: *matrices* y *bytes*.

Para grabar matrices tenemos que incluir la cláusula **DATA** en la orden **SAVE**:

SAVE *nombre-de-fichero* **DATA** *nombre-de-matriz***()**

donde *nombre-de-fichero* es el nombre con el que la información quedará grabada en el disco y está sometido a las mismas reglas que los nombres de los ficheros en que almacenamos programas.

El *nombre-de-matriz* especifica la matriz de BASIC que queremos grabar, así que tiene que ser una sola letra (o una letra seguida de **$**). No olvide poner los paréntesis **( )** tras el nombre de la matriz.

Es necesario distinguir bien los papeles que desempeñan el *nombre-de-fichero* y el *nombre-de-matriz*. Por ejemplo, con la orden

SAVE "vargas" DATA b()

lo que hacemos es que **SAVE** lea la matriz **b** en la memoria y la almacene en el disco con el nombre de **VARGAS**.

La orden

LOAD "vargas" DATA b()

averigua si hay espacio en la memoria para la matriz; si lo hay, borra la matriz que pueda existir en la memoria con el nombre **b**, lee el fichero **VARGAS** en el disco y carga la matriz que encuentra en él, dándole el nombre **b**.

No se puede usar **MERGE** para cargar matrices.

También podemos grabar matrices de caracteres (literales), exactamente de la misma forma que las numéricas. Sin embargo, cuando cargamos una matriz literal, +3 BASIC borra, no sólo la matriz literal, sino también la variable literal que pueda existir en la memoria con el mismo nombre.

Cuando se está trabajando con grandes cantidades de datos, puede interesar grabarlos en la unidad M (con **SAVE...DATA**) para luego cargarlos (con **LOAD...DATA**). Una vez grabada la información en la unidad M, el espacio que ocupaba la matriz en la memoria puede ser reutilizado. La transferencia de datos entre la memoria de BASIC y la unidad M tiene la ventaja de ser muy rápida.

El almacenamiento de bytes se aplica a bloques de información cualesquiera, con independencia de la naturaleza de esa información (podría ser una imagen de la pantalla, unos gráficos de usuario, etc.). Este tipo de grabación se realiza incluyendo la cláusula **CODE** en la orden **SAVE**; por ejemplo,

SAVE "imagen.bin" CODE 16384,6912

La unidad de almacenamiento en la memoria es el *byte* (un número entero comprendido entre 0 y 255); cada byte de la memoria está identificado por una *dirección* (que es un nú-

mero entero comprendido entre 0 y 65535). El primer número que se pone después de **CODE** es la dirección del primer byte que debe ser grabado en el disco; el segundo número es la cantidad de bytes que deben ser grabados. En nuestro caso, 16384 es la dirección del primer byte que será transferido al fichero (el cual contendrá la descripción de la imagen de la pantalla); 6912 es el número de bytes que habrá en el fichero, los necesarios para describir la pantalla completa. Pruebe la anterior orden **SAVE...CODE**. (No tiene por qué grabar los bytes precisamente con el nombre **IMAGEN.BIN**; nosotros hemos elegido este nombre para que nos recuerde cuál es la naturaleza de la información grabada.)

Para cargar esta información se da la orden:

LOAD "imagen.bin" CODE

También podemos poner parámetros detrás de **CODE**:

LOAD *nombre-de-fichero* **CODE** *comienzo,longitud*

Aquí *longitud* se utiliza como medida de seguridad: cuando el ordenador se pone a cargar los bytes, comprueba la *longitud* y rehusa cargarlos si hay más de los especificados; de esta forma se previene el riesgo de sobreescritura accidental de una zona de la memoria que interese preservar. En tal caso, +3 BASIC emite el mensaje de error '**ERROR DE LONGITUD**'. (Si se produce el error equivalente al cargar bytes desde la cinta en 48 BASIC, el mensaje es '**ERROR CARGANDO CINTA**'.)

Si omitimos el parámetro *longitud*, el ordenador carga todos los bytes que encuentra en el fichero, cualquiera que sea su número.

El parámetro *comienzo* especifica la dirección de la memoria en la que debe ser cargado el primer byte, que puede ser diferente de la dirección desde la que fue grabado. Si ambas direcciones son iguales, podemos omitir este parámetro en la sentencia **LOAD**.

CODE 16384,6912 es un área de la memoria (imagen de pantalla) tan útil, que BASIC dispone de una función especial, **SCREEN$**, para representarla. Así, para grabar y cargar la pantalla se puede usar las órdenes:

SAVE "imagen.bin" SCREEN$

LOAD "imagen.bin" SCREEN$

Copias de seguridad automáticas

Supongamos que tenemos en el disco varios ficheros y que grabamos un programa (o cualquier otra información) con un nombre ya utilizado. ¿Causará esto algún conflicto?

Cada vez que intentamos grabar un programa, el sistema de almacenamiento en disco comprueba si ya existe un fichero con el nombre especificado. De ser así, lo primero que hace es cambiarle el nombre al fichero antiguo para no tener que destruirlo. En realidad, el cambio de nombre sólo afecta al tipo, que pasa a ser **.BAK**.

Ahora bien, si ya existe una versión **.BAK**, ésta sí es destruida. Esto quiere decir que en el disco puede haber, como máximo, dos versiones del fichero: la recién grabada y la inmediatamente anterior (que es la que lleva el tipo **.BAK**). De esta forma, si grabamos varias veces distintas versiones de un fichero, siempre dispondremos de una copia de seguridad a la que retroceder en caso de que estropeemos la versión más moderna. Si esto llega a ocurrir, todo lo que necesitamos es borrar la versión más reciente y cambiarle el nombre a la anterior (**.BAK**). En el apartado siguiente veremos cómo hacerlo. Antes de continuar, grabe una vez más el programa de los cuadrados con la orden:

```
SAVE "m:cuadrado"
```

Borrado y cambio de nombre de los ficheros

La orden con la que podemos borrar ficheros es **ERASE**. Esta palabra debe ir seguida del nombre de fichero que queramos borrar. Al igual que en **CAT**, podemos usar los caracteres polivalentes * y ? para construir una plantilla que especifique un grupo de ficheros. Si sólo especificamos un fichero, **ERASE** lo borrará inmediatamente, sin pedir confirmación. En cambio, si especificamos un grupo de ficheros (usando * o ?), BASIC nos pregunta si de verdad queremos borrar esos ficheros. En caso afirmativo, se pulsa [S] para permitir que el proceso de borrado continúe. Pulsando [N] se abandona la orden **ERASE**.

Por ejemplo, para borrar el fichero **CARAMELO.BAS** de la unidad M podemos dar la orden:

```
ERASE "m:caramelo.bas"
```

(Si la unidad M es la implícita, no es necesario escribir su letra como prefijo del nombre de fichero. Sin embargo, dado que los efectos de la orden **ERASE** son irreversibles, no se pierde nada por hacerlo.)

Para borrar todos los ficheros de la unidad B (si estuviera conectada) daríamos la orden:

```
ERASE "b:*.*"
```

BASIC pide entonces confirmación a través del siguiente mensaje:

```
¿Borrar b:*.* (S/N)?
```

En respuesta a esta pregunta sólo debemos pulsar [S] en el caso de que efectivamente estemos dispuestos a borrar todos esos ficheros.

Si **ERASE** no encuentra ningún fichero que encaje en la especificación, emite el mensaje **'FICHERO NO ENCONTRADO'**.

Observe que si la especificación consiste solamente en la letra de la unidad (v.g., **ERASE "m:"**), la orden borra **todos los ficheros** de esa unidad **sin pedir confirmación**. Por consiguiente, antes de dar esta orden piense bien qué es exactamente lo que quiere borrar.

El proceso de borrado se interrumpe en cuanto **ERASE** detecta que un fichero (o el disco) está 'protegido contra escritura'.

El disco que hemos estado usando contiene varias copias del mismo programa grabadas con diferentes nombres. Esto representa un despilfarro de espacio de disco. Lo que tendríamos que hacer sería borrar todos los ficheros que no necesitemos; es decir, todos menos **CUADRADO**. En general, no hay una forma directa de borrar todos los ficheros menos uno. Puesto que algunos ficheros tienen alguna letra en común, podemos ahorrarnos trabajo si construimos plantillas con * y ?. (Piense qué plantillas serían necesarias para borrar todos los ficheros menos **CUADRADO** con el mínimo número de órdenes **ERASE** posible.)

Una vez grabado un fichero, podemos cambiarle el nombre mediante la orden **MOVE**. Por ejemplo, si tenemos en la unidad M el fichero **CUADRADO** y queremos cambiarle el nombre a **BLOQUES**, podemos hacerlo con las siguientes órdenes:

```
SAVE "m:cuadrado"      (por si no estuviera CUADRADO en la unidad M)
MOVE "m:cuadrado" TO "m:bloques"
CAT  "m:"
```

Supongamos que hemos grabado el programa **BINGO** y que, después de modificarlo y grabarlo con el mismo nombre, nos damos cuenta de que lo hemos destrozado completamente y necesitamos recuperar la versión anterior. Las órdenes que necesitamos son:

```
ERASE "bingo"
MOVE "bingo.bak" TO "bingo"
```

MOVE no admite caracteres polivalentes (* y ?) cuando la usamos para cambiar el nombre de los ficheros.

Por otra parte, **MOVE** tiene en cuenta cuál es la unidad implícita, de modo que no es necesario especificar la letra de la unidad si el fichero cuyo nombre vamos a cambiar está en la unidad implícita.

No se puede usar **MOVE** para trasladar un fichero de una unidad a otra distinta. Por consiguiente, una orden tal como:

```
MOVE "m:bingo" TO "a:bonga"
```

fracasaría y provocaría el mensaje de error **'SINTAXIS INCORRECTA'**. El procedimiento para trasladar un fichero a otra unidad consiste en copiarlo con **COPY** (véase más adelante) y después borrar el original con **ERASE**.

Atributos de los ficheros

MOVE no sólo sirve para cambiar el nombre de los ficheros, sino también para cambiarles los *atributos*.

Los atributos son elementos de información que están asociados a cada fichero y que determinan si el fichero tiene o no tiene ciertas propiedades.

Son tres los atributos que podemos modificar con **MOVE**. El más útil es la 'protección contra escritura'. Cuando este atributo se encuentra activado, no es posible borrar el fichero (ni grabar otro con el mismo nombre en el mismo disco). Funciona igual que el orificio de protección contra escritura de los discos, pero afecta a ficheros individuales. Sin embargo, no proporciona ninguna protección contra **FORMAT**, que borra todos los ficheros, incluso los protegidos por este atributo. Para activar el atributo de protección de un fichero necesitamos una orden del siguiente estilo:

```
MOVE "cuadrado" TO "+p"
```

(La letra '**p**' es abreviatura de '**p**rotección'.) Si luego intentamos borrar el fichero:

```
ERASE "cuadrado"
```

la orden **ERASE** fracasa y provoca el mensaje de error '**FICHERO SOLO LECTURA**'. Para desactivar la protección ejecutamos la orden:

```
MOVE "cuadrado" TO "-p"
```

y entonces el fichero ya puede ser borrado.

En todas estas versiones de la orden **MOVE**, el signo + significa 'activar', mientras que − significa 'desactivar'.

Cuando usamos **MOVE** para cambiar los atributos de los ficheros, sí admite la especificación de grupos de ficheros mediante plantillas construidas a base de * y ?. Por ejemplo, para proteger todos los ficheros de la unidad M daríamos la orden:

```
MOVE "m:*.*" TO "+p"
```

Como siempre, podemos omitir la letra de la unidad cuando queramos que la orden afecte a la unidad implícita.

No se provoca un error por el hecho de que activemos un atributo que ya está activado; el atributo sencillamente queda como estaba. Lo mismo se puede decir acerca de la desactivación.

El segundo atributo es el de 'estado de sistema'. En el **+2A** este atributo no tiene demasiado interés; ha sido incluido sobre todo por compatibilidad con el sistema operativo CP/M. El atributo de estado de sistema se especifica en las órdenes **MOVE** mediante +**S** o −**S**.

Si un fichero tiene activado este atributo, es un fichero 'de sistema' y su nombre no aparece en los listados obtenidos con **CAT**. Por el contrario, si este atributo está desactivado, el fichero es 'de directorio' y su nombre sí sale en los listados de **CAT**.

Por otra parte, si pedimos un catálogo ampliado (con **CAT EXP**), la lista incluye también los ficheros de sistema (tras cuyos nombres aparecen las letras '**SYS**'). Además, si hay algún fichero protegido, su nombre estará señalado con las letras '**PROT**'.

El atributo de estado de sistema tiene utilidad, por ejemplo, cuando el disco contiene muchos ficheros y queremos ocultar algunos para que los catálogos no sean tan confusos.

En un mismo disco no puede haber dos ficheros con el mismo nombre, ni siquiera en el caso de que sus atributos de estado de sistema sean diferentes. Por consiguiente, si grabamos un fichero y ya existe en el disco otro fichero con el mismo nombre (pero está oculto para **CAT**), el resultado es que borraremos este último.

El último atributo que podemos controlar es el de 'archivo'. En los catálogos ampliados, los ficheros de archivo están identificados por las letras '**ARC**'. Este atributo se especifica en las órdenes **MOVE** mediante **+A** o **−A**. En el **+2A** este atributo no tiene ninguna utilidad; ha sido incluido sobre todo por compatibilidad con el sistema operativo CP/M.

Veamos unas cuantas órdenes de control de atributos (trate usted de predecir cuáles serán sus efectos):

```
MOVE "*.*" TO "+p"
MOVE "*.bas" TO "-s"
MOVE "c???.*" TO "+a"
MOVE "m:*.*" TO "-p"
```

Si intentamos usar una letra de atributo que no sea **A**, **S** ni **P**, o si la especificación contiene más de dos caracteres, provocaremos el error '**ATRIBUTO NO VALIDO**'.

Copia de ficheros

Con gran frecuencia tendremos necesidad de copiar ficheros (por ejemplo, para darle a un amigo copias de nuestros programas, o para colocar los ficheros en la unidad M, en la que el acceso es más rápido). Si tenemos conectada la unidad de disco externa, la orden **COPY** nos permite copiar ficheros de una unidad a otra, e incluso copiar discos enteros. La versión más sencilla tiene la siguiente forma:

```
COPY "a:banco" TO "m:"
```

que significa 'leer el fichero **BANCO**, que se encuentra en la unidad A, y grabar una copia en la unidad M'. Como no hemos especificado ningún nombre de fichero para la copia, la orden conserva el nombre original. (Si no está conectada la unidad externa, esta orden provoca el error '**UNIDAD NO ENCONTRADA**'; en realidad, **COPY** no es muy útil en los sistemas que sólo disponen del disco de RAM.)

El nombre que ponemos antes de la palabra **TO** es el nombre del fichero de *origen*; el que ponemos después de **TO** es el nombre del fichero de *destino*.

La orden:

```
COPY "banco" TO "bingo"
```

lee el fichero **BANCO** en la unidad implícita y deposita la copia con el nombre **BINGO** en la misma unidad. Al final del proceso, ambos ficheros contendrán exactamente la misma información.

Observe que cuando la unidad de origen es la misma que la de destino, es obligatorio especificar el nombre del fichero de destino.

Si intentamos copiar un fichero en la misma unidad y con el mismo nombre, provocaremos el error **'YA EXISTE EL FICHERO'** (o quizá **'FICHERO YA EN USO'**).

El nombre de origen puede ser una plantilla construida con los símbolos * y ? para especificar un grupo de ficheros. En tal caso, para el destino sólo se puede especificar la letra de la unidad. Por ejemplo,

```
COPY "a:*.ovl" TO "m:"
```

copia todos los ficheros que encuentra en la unidad A con el tipo **OVL** y deposita las copias en la unidad M, conservando los nombres. En cambio, la orden:

```
COPY "a:*.bas" TO "m:*.bin"
```

fracasa y provoca el error **'DESTINO POLIVALENTE'**.

La orden **COPY** no copia los atributos de los ficheros. Por lo tanto, si queremos que las copias tengan activado algún atributo, tendremos que ejecutar las órdenes **MOVE** necesarias.

COPY emite una lista con los ficheros que va copiando, en dos columnas. De esta forma es muy fácil comprobar que los ficheros que encajan en una plantilla son precisamente los que queríamos copiar.

Al final del proceso, **COPY** da un informe en el que dice cuántos ficheros ha copiado. Si la especificación del origen era una plantilla, este informe sirve para comprobar que hemos copiado los ficheros que queríamos.

Cuando interese usar la orden **COPY** desde dentro de un programa y al mismo tiempo evitar que el informe generado por **COPY** destruya el texto o los gráficos generados por el propio programa, se puede inhibir el listado incluyendo

```
POKE 23739,87: POKE 23740,10
```

antes de la orden **COPY** y

```
POKE 23739,244: POKE 23740,9
```

después de ella. Por ejemplo,

```
10 POKE 23739,87: POKE 23740,10
20 COPY "a:disk.*" TO "m:"
30 POKE 23739,244: POKE 23740,9
```

Hay una forma especial de la orden **COPY**:

COPY "a:" TO "b:"

que realiza una copia 'sector por sector' del disco de la unidad A hacia el disco de la unidad B (el cual tiene que haber sido formateado previamente). En este caso, el contenido antiguo del disco de la unidad B se pierde. Por otra parte, si los ficheros que hay en el disco de origen no son muchos, es más rápido copiarlos con:

COPY "a:*.*" TO "b:"

Aunque sólo tengamos conectada una unidad de disco externa, podemos usarla como si fuera la unidad A y la B. Supongamos que éste es el caso y que queremos copiar de un disco a otro los dos únicos ficheros de tipo **BAS** que tenemos. El procedimiento es el siguiente: introduzca en la (única) unidad el disco de origen y dé la orden

COPY "a:*.bas" TO "b:"

En cuanto el **+2A** lee una parte del primer fichero **.BAS**, emite el mensaje:

Introduzca en la unidad el disco
para B y luego pulse una tecla

Haga lo que le ha pedido el mensaje. El **+2A** escribe la información en la 'unidad B' y luego le pide que:

Introduzca en la unidad el disco
para A y luego pulse una tecla

Este proceso de intercambio de discos continúa hasta que todos los ficheros han quedado copiados. La orden **COPY** trata de utilizar el espacio que queda libre en la unidad M; cuanto más espacio pueda conseguir, menos cambios de disco tendremos que hacer. Por eso conviene, si es posible, borrar la unidad M antes de ponerse a copiar gran número de ficheros.

Además de para copiar ficheros de una unidad a otra, **COPY** sirve para copiar ficheros hacia la pantalla o hacia la impresora (si está conectada). Por ejemplo, la orden

COPY "letras.txt" TO SCREEN$

muestra en la pantalla el contenido del fichero **LETRAS.TXT** (de la unidad implícita). Esta orden filtra todos los códigos de control, excepto el 'retorno del carro'. No es adecuada para examinar ficheros de programa de BASIC, a causa de los numerosos códigos de control que contienen. Sí sirve, en cambio para inspeccionar ficheros de texto ASCII, tales como los creados con un procesador de texto.

La orden

COPY "letras.txt" TO LPRINT

es similar a la anterior, pero envía el contenido del fichero a la impresora y **no** suprime los códigos de control. Si hemos dirigido la salida de impresora hacia el interfaz serie y hemos prohibido la conversión de códigos (con **FORMAT LPRINT "R";"U"**), esta orden nos permite 'exportar' ficheros hacia otros ordenadores. Esto no tiene ninguna utilidad cuando los ficheros son programas de BASIC.

Los usuarios que escriban programas en código de máquina pueden preferir desarrollarlos en una máquina más potente (por ejemplo, en un AMSTRAD PCW). Una vez terminado el programa, lo más probable es que el **+2A** no reconozca el fichero, ya que +3 BASIC siempre espera encontrar una *cabecera* de 128 bytes al principio del fichero, en la que debe estar cierta información requerida por la orden **LOAD**. Sin embargo, si tenemos un fichero binario en un disco adecuado para el **+2A**, podemos añadirle la cabecera correcta mediante una orden tal como:

```
COPY "juego.com" TO SPECTRUM FORMAT
```

Esta orden crea un nuevo fichero en la misma unidad, con el mismo nombre y con el tipo **.HED**. (En este ejemplo, crearía el fichero **JUEGO.HED** en la unidad implícita.)

Evidentemente, esta orden sólo tiene interés cuando la aplicamos a ficheros de programa de código de máquina. La longitud del fichero queda anotada en la cabecera, y también el hecho de que se trata de un fichero de tipo **CODE**. Sin embargo, BASIC no tiene forma de saber en qué dirección de memoria tiene que cargar el fichero; por lo tanto, la dirección de carga deberá ser especificada en la orden **LOAD...CODE**. Por ejemplo, si el programa ha sido ensamblado tomando como dirección inicial 7000h (donde la 'h' indica que es un número hexadecimal) o 28672 decimal, el fichero con cabecera debe ser cargado con:

```
LOAD "juego.hed" CODE 28672
```

Puesto que los ficheros de pantalla son también ficheros '**CODE**', esta técnica sirve para 'importar' pantallas diseñadas en otra máquina. (Esto no servirá de nada si las pantallas no han sido diseñadas teniendo en cuenta el tamaño y el formato de la pantalla del **+2A**.)

El disco de RAM

Posiblemente se preguntará usted de qué sirve almacenar información en el disco de RAM (unidad M) cuando sabemos que su contenido se pierde en cuanto apagamos el ordenador. Quizá la aplicación más obvia de la unidad M sea el almacenamiento de porciones (o 'rutinas') de un programa de BASIC para mezclarlas (con **MERGE "m:*nombre-de-fichero*"**) sucesivamente con un programa más pequeño. Esto hace posible tener en la memoria un programa de cerca de 90K (para ello la estructura del programa tiene que haber sido muy bien diseñada).

En la práctica, suponiendo que dispongamos de una unidad externa, podemos almacenar las diferentes rutinas en un disquete y copiarlas hacia la unidad M con la orden **COPY**.

La ventaja de tener las rutinas en la unidad M está en que el acceso a ella es mucho más rápido que a las unidades mecánicas (A y B). En contrapartida, las unidades mecánicas tienen una capacidad superior. La solución ideal será en muchos casos diseñar los programas de forma que utilicen tanto los disquetes como la unidad M.

Una de las aplicaciones más atractivas del disco de RAM es la *animación* de imágenes. Un programa de BASIC, de por sí relativamente lento, puede definir una serie de imágenes y almacenarlas en la memoria. Éstas pueden ser luego transferidas a la pantalla a gran velocidad. El siguiente programa da una idea de lo que se puede lograr con esta técnica (sin duda, usted puede hacer algo mejor):

```
10 INK 5: PAPER 0: BORDER 0: CLS
20 FOR f=1 TO 10
30 CIRCLE f*20,150,f
40 SAVE "m:balon"+ STR$ (f) CODE 16384,2048
50 CLS
60 NEXT f
70 FOR f=1 TO 10
80 LOAD "m:balon"+ STR$ (f) CODE
90 NEXT f
100 BEEP 0.01, 0.01
110 FOR f=9 TO 2 STEP -1
120 LOAD "m:balon"+ STR$ (f) CODE
130 NEXT f
140 BEEP 0.01, 0.01
150 GO TO 70
```

(Antes de ejecutar el programa compruebe si la unidad M está vacía; si no lo está, dé una orden **ERASE "m:*.*"** y pulse [S] para responder a la pregunta.)

En la línea 40 de este programa, los dos números que siguen a **CODE** son la dirección de memoria donde empieza la pantalla (**16384**) y la longitud del tercio superior de ésta (**2048**). Al grabar y cargar solamente ese trozo de la pantalla se mejora la velocidad global.

Operaciones con la cinta

Gran parte de lo que hemos dicho acerca de **LOAD**, **SAVE** y **MERGE** en los apartados anteriores es válido para el sistema de almacenamiento en cinta. Sin embargo, las órdenes **FORMAT**, **COPY**, **MOVE**, **CAT** y **ERASE** no son aplicables a la cinta (si bien se puede usar una forma especial de **CAT**).

Cuando encendemos el **+2A**, la unidad que queda seleccionada automáticamente como 'unidad por defecto' es la A (suponiendo que esté conectada la unidad externa). Esto representa que todas las órdenes **CAT**, **ERASE**, **LOAD**, **SAVE**, etc. que ejecutemos sin es-

pecificar letra de unidad afectarán a la unidad A. Como usted recordará, para cambiar de unidad implícita podemos dar una de las siguientes órdenes:

LOAD "*letra-de-unidad*:"

o

SAVE "*letra-de-unidad*:"

donde *letra-de-unidad* puede ser **A**, **B** o **M**. (Observe que la *letra-de-unidad* siempre va seguida del signo de dos puntos.) De hecho, también podemos usar la **T** como letra de unidad, pero sólo en estas formas especiales de **LOAD** y **SAVE**. Por ejemplo, si damos la orden:

LOAD "t:"

todas las operaciones de carga (con **LOAD** o **MERGE**) siguientes leerán en la cinta (hasta que volvamos a seleccionar el disco como unidad implícita; por ejemplo, con **LOAD "a:"**). Análogamente, si ejecutamos la orden:

SAVE "t:"

todas las operaciones de grabación (con **SAVE**) siguientes escribirán en la cinta (hasta que volvamos a seleccionar el disco como unidad implícita; por ejemplo, con **SAVE "a:"**).

Si no está conectada la unidad externa, el ordenador ya ha seleccionado **"t:"** (o sea, el magnetófono) como unidad implícita para **LOAD**, **VERIFY** y **MERGE**, por lo que no será necesario dar las órdenes anteriores.

La especificación de **T** como letra de unidad sólo afecta a las órdenes **LOAD**, **MERGE** y **SAVE**, no a **MOVE**, **COPY**, **CAT** y **ERASE**, para las cuales la unidad implícita sigue siendo la misma (ya que estas órdenes no son aplicables a la cinta).

Quizá todo esto suene más complicado de lo que es en realidad, así que vamos a aclararlo con unos ejemplos. Si suponemos que acabamos de encender (o de reinicializar) el **+2A**, la unidad implícita para todas las operaciones será la cinta (a menos que tengamos conectada la unidad externa, porque en tal caso sería la A). Si ahora ejecutamos la orden:

SAVE "m:"

la nueva unidad implícita para **todas** las operaciones de disco (excepto **LOAD** y **MERGE** cuando no existe la unidad externa) pasa a ser la M.

Si ahora damos la orden:

LOAD "b:"

queda seleccionada la unidad B (si existe) como unidad implícita para todas las operaciones.

Es importante observar que tanto **LOAD** como **SAVE** seguidas de **"A:"**, **"B:"** o **"M:"** seleccionan la unidad implícita para todas las órdenes de disco futuras (**LOAD**, **SAVE**,

MERGE, CAT, ERASE, etc.). En cambio, **LOAD "t:"** sólo afectará a las operaciones **LOAD** y **MERGE** futuras, y **SAVE "t:"** sólo afectará a las órdenes **SAVE** futuras.

Si ahora damos la orden:

```
SAVE "t:"
```

las operaciones **SAVE** serán desviadas hacia la cinta, pero la unidad B seguirá siendo la implícita para todas las demás. Si luego damos la orden:

```
LOAD "t:"
```

ésta desvía las futuras órdenes **LOAD** y **MERGE** hacia la cinta, pero mantiene la unidad B como unidad implícita para todas las órdenes específicas de disco.

Si en esta situación damos una orden:

```
SAVE "a:"
```

todas las operaciones de disco son redirigidas hacia la unidad A, excepto **LOAD** y **MERGE**, que siguen desviadas hacia la cinta.

Como ejercicio práctico vamos a grabar el programa de los cuadrados en una cinta. Reinicialice el ordenador, seleccione la opción **+3 BASIC** y transcriba este programa:

```
10 POKE 22527+ RND *704, RND *127
20 GO TO 10
```

Si tiene conectada la unidad externa, puede cargar el programa con la orden:

```
LOAD "cuadrado"
```

Éste es el programa que vamos a grabar en la cinta. (Dicho sea de paso, aunque vale cualquier cinta de cassette, son preferibles las de bajo nivel de ruido.)

Escriba ahora lo siguiente:

```
SAVE "t:"
SAVE "cuadrado"
```

(**SAVE "t:"** sólo es necesaria en los ordenadores que tienen conectada la unidad externa.) Estas órdenes grabarán el programa en la cinta con el nombre de **'CUADRADO'**. Los nombres de fichero de cinta admiten hasta 10 caracteres; se puede usar caracteres cualesquiera, e incluso espacios.

Obedezca el mensaje:

```
PREPARE LA CINTA Y PULSE INTRO
```

Seguramente recordará (porque lo dijimos en el Capítulo 6) que los colores del borde de la pantalla cambian para indicar que el ordenador está grabando en la cinta.

Cuando el **+2A** haya terminado de grabar (mensaje **0 OK**), pulse el botón de parada del magnetófono.

Después de grabar un programa, y antes de borrarlo de la memoria del ordenador, debemos comprobar que el proceso de grabación ha funcionado correctamente. Para ello se utiliza la orden **VERIFY**, descrita en el Capítulo 6.

Supongamos que usted ha grabado el programa y lo ha verificado con éxito. Cárguelo en la memoria con la orden:

LOAD "cuadrado"

(Puesto que la verificación tuvo éxito, la carga no debería dar ningún problema.)

La orden **MERGE** funciona de forma parecida a la descrita antes en relación con los ficheros de disco. La principal diferencia consiste en que ahora podemos usar **MERGE ""** con el significado de 'mezcla el siguiente fichero que encuentres en la cinta'. (Los nombres de fichero de cinta especificados tras **MERGE** pueden contener hasta 10 caracteres cualesquiera, e incluso espacios.)

Si usted tiene programas de BASIC grabados en cinta, puede transferirlos a un disco (a condición de que esté conectada la unidad externa). El procedimiento es muy sencillo. Dé las órdenes:

LOAD "t:"
SAVE "a:"

para dirigir las operaciones de carga hacia la cinta y las de grabación hacia el disco. Después, para cada programa de BASIC de la cinta, cárguelo con:

LOAD ""

(Esta orden significa 'carga el primer programa que encuentres en la cinta'.) Una vez cargado el programa, elija un nombre adecuado y grábelo en el disco con:

SAVE "*nombre*"

(Recuerde que los ficheros de disco necesitan un nombre no vacío.)

Si un programa de cinta ha sido grabado con la cláusula **LINE**, al cargarlo el ordenador lo pondrá en marcha inmediatamente. Como no es esto lo que queremos, cargue el programa con **MERGE ""** en vez de hacerlo con **LOAD ""**.

Si tiene grabadas matrices de datos (numéricas o literales), cárguelas con **LOAD** y grábelas en el disco con **SAVE**.

Los únicos ficheros que pueden dar complicaciones a la hora de transferirlos de cinta a disco son los de tipo **CODE** (y **SCREEN$**). Para transferir un fichero de este tipo hace falta conocer:

1. La dirección de memoria desde la que se lo grabó.
2. Su longitud (en número de bytes).

Catálogo de la cinta

Para obtener éstos y otros datos disponemos de una forma especial de la orden **CAT**, que consiste en usar '**T:**' como especificación de fichero.

Si damos la orden:

```
CAT "t:"
```

el **+2A** espera hasta que pongamos la cinta en marcha (esta operación puede ser abandonada en cualquier momento pulsando [BREAK]). Cada vez que el ordenador encuentra en la cinta una cabecera de fichero, muestra la información leída en la misma forma en que había sido grabada. El primer dato es un nombre de fichero entre comillas. Los siguientes dependen de la naturaleza de la información que contenga el fichero. Si se trata de un programa de BASIC, **CAT** muestra la palabra **(BASIC)**, precedida de **LINE** y un número de línea en caso de que se hubiera usado esta opción al grabar el programa. Si el fichero contiene datos, **CAT** escribe la palabra **DATA** y a continuación el nombre de la matriz. Si el fichero ha sido grabado con la opción de CODE (o **SCREEN$**, que equivale a **CODE 16384,6912**), **CAT** escribe la palabra **CODE** seguida de la dirección inicial y la longitud especificadas en la grabación. Así pues, **CAT "t:"** podría producir una lista del siguiente estilo:

```
"simple       " (BASIC)
"autoejec     " LINE 10 (BASIC)
"cifras       " DATA f()
"palabras     " DATA c$()
"binario      " CODE 30000,12345
"dibujo       " CODE 16384,6912
```

El último de estos ficheros podría haber sido grabado con:

```
SAVE "dibujo" SCREEN$
```

La salida de esta forma de **CAT** también puede ser dirigida hacia la impresora especificando el canal número 3:

```
CAT #3,"t:"
```

(Los canales están explicados en la Sección 22 de este capítulo.) Esto sólo es posible si se tiene conectada una impresora. (Para abandonar se pulsa [BREAK].)

A la vista de la información proporcionada por **CAT "t:"** podemos cargar (con **MERGE ""**) el programa de ejecución automática y luego grabarlo en disco con la opción **LINE** y con el número de línea que podemos ver en la lista, de modo que la versión de disco sea también de ejecución automática:

SAVE *nombre* **LINE** *número*

Por otra parte, para transferir de cinta a disco un fichero de tipo **CODE** tomamos nota de la dirección inicial y la longitud mostradas por **CAT "t:"**, damos la orden:

CLEAR *dirección-inicial*−**1**

y cargamos el fichero (después de hacer retroceder la cinta) con:

LOAD "" CODE

Cuando aparezca el mensaje **0 OK** podemos grabar el fichero en disco con la orden:

SAVE *nombre* **CODE** *dirección-inicial,longitud*

(Esta técnica sólo debe ser aplicada a la transferencia de los ficheros propios. Copiar un programa comercial por este procedimiento puede constituir un fraude y una violación de copyright.)

Hay varias razones por las que este método puede no funcionar:

1. Cuando la información, una vez cargada, se superpone a las variables del sistema [que se encuentran en el margen de 23926 (5B00h) a 23755 (5CC6h)]. Este límite superior es variable; su valor está en la variable PROG (v. Sección 25).
2. Si se intenta cargar un fichero que no tenga cabecera (o que haya sido protegido de alguna forma) es posible que sus datos ni siquiera aparezcan en la lista generada por **CAT "t:"**. Desde luego, no será posible cargarlo con la orden **LOAD** de BASIC.
3. Si el fichero es tan largo que se extiende desde la zona de la imagen de pantalla hasta el final de la memoria, será posible cargarlo, pero se perderá el control de la máquina porque la 'pila' de BASIC habrá sido invadida.

Ejercicio

1. Practique todas las operaciones descritas en esta sección hasta que considere que puede manejar con soltura los ficheros, tanto en el disco de RAM como en la cinta y en la unidad de disquete (si la tiene conectada).

Sección 21
Operaciones con la impresora

Temas tratados:

Impresoras 'serie'
Impresoras 'paralelo'
LPRINT, LLIST
FORMAT
COPY

El **+2A** tiene incorporada una puerta paralelo Centronics de 8 bits, una puerta serie RS232 y el software necesario para controlar una impresora. Estos recursos sólo pueden ser utilizados en modo +3 **BASIC**.

La impresora debe tener un interfaz de tipo 'paralelo Centronics' o de tipo 'serie RS232'; además, para que pueda reproducir las imágenes de pantalla, debe tener *modo gráfico de imagen de bits de cuádruple densidad compatible Epson* (ESC **"L"** *n n*).

Asegúrese de que dispone del cable correcto para conectar la impresora con el **+2A**; en caso de duda, consulte con su distribuidor.

En el Capítulo 10, 'Periféricos para el **+2A**', explicaremos qué impresoras y qué cables son adecuados para este ordenador y describiremos las conexiones de los zócalos **IMPRESORA** y **RS232**.

Impresoras 'paralelo'

Cuando encendemos el **+2A**, éste supone que la impresora, si existe, está conectada en el zócalo **IMPRESORA**, que es la puerta 'paralelo' del ordenador. La conexión entre el ordenador y la impresora es relativamente directa; la única precaución que hay que tomar es no insertar boca abajo el lado del cable que va al ordenador.

Una vez instalado el cable, una orden tal como

```
LPRINT "¡Hola!"
```

debe producir algún resultado en la impresora; de no ser así, compruebe la conexión y cerciórese de que la impresora está 'en línea'.

Cuando haya conseguido imprimir alguna palabra, puede pasar a la sección 'Control de la impresora'.

Impresoras 'serie'

Las conexiones entre una impresora serie y el **+2A** dependen del modelo concreto de impresora. Asegúrese de que el cable que le proporciona su distribuidor es el adecuado para conectar su impresora con el **+2A**.

Las impresoras serie deben ser conectadas al zócalo **RS232** del ordenador, que está descrito en el Capítulo 10, 'Periféricos para el **+2A**'.

El interfaz serie del **+2A** utiliza el denominado *control de flujo por hardware*. Este sistema consiste en que el interfaz no envía caracteres a la impresora mientras no haya recibido de ésta ciertas señales que autorizan la transmisión. Por consiguiente, es muy importante que el cable haga las conexiones correctas, no sólo en las patillas de entrada y salida de datos, sino también en las líneas de control. Si la impresora no maneja el sistema de control de flujo por hardware, se debe interconectar las patillas 4 y 5 del zócalo **RS232**. La desventaja de no usar este sistema de control es la posibilidad de perder algún carácter cuando se transmita grandes bloques de datos a gran velocidad.

Para que el **+2A** y la impresora puedan comunicarse el uno con la otra, ambos deben trabajar con la misma *velocidad de transmisión*, que es, como su nombre indica, la velocidad a la que se transfiere los datos del ordenador a la impresora. Aunque la velocidad de transmisión pueda ser seleccionada en la impresora, probablemente será más fácil cambiarla, si es necesario, en el ordenador. En algún lugar del manual de la impresora estará especificada la velocidad de transmisión; averigüe cuál es y selecciónela en el **+2A** con la orden:

FORMAT LINE *velocidad-de-transmisión*

Por ejemplo,

FORMAT LINE 300

(No será necesario hacer este ajuste si la impresora funciona normalmente a 9600 baudios, pues ésta es la velocidad que el **+2A** adopta por defecto.)

Puesto que inicialmente el **+2A** supone que la impresora está conectada en la puerta paralelo, tenemos que avisarle de que nuestra impresora es de tipo serie y está conectada en el zócalo **RS232**. Para ello debemos dar la orden:

FORMAT LPRINT "R"

La orden con la que redirigimos la salida de impresora hacia la puerta paralelo es:

FORMAT LPRINT "C"

Control de la impresora

Una vez preparada la conexión, ya podemos empezar a usar las tres órdenes de BASIC que controlan la impresora. Las dos primeras, **LPRINT** y **LLIST**, son análogas a **PRINT** y **LIST**, salvo que dirigen la salida a la impresora en lugar de a la pantalla. La opción **Imprimir** del menú de edición de +3 BASIC produce el mismo efecto que **LLIST**, pero la hemos incluido para proporcionar una forma más directa y fácil de obtener un listado.

Pruebe este programa:

```
10 LPRINT "Este programa "'
20 LLIST 40
30 LPRINT '"imprime el juego de caracteres: "'
40 FOR n=32 TO 255
50 LPRINT CHR$ n;
60 NEXT n
70 LPRINT
```

Es importante observar que **LPRINT** y **LLIST** tienen buen cuidado de filtrar todos los códigos de color (y sus parámetros) que encuentran antes de imprimir o listar cualquier cosa. Tales códigos de color son una «reliquia» del Spectrum de 48K; cuando se los incluía en una cadena ajustaban **INK** (tinta), **PAPER** (papel), etc. En general las impresoras utilizan esos códigos para cosas completamente diferentes, como activar o desactivar cursiva, subrayado, etc., así que sería muy peligroso enviarles estos códigos de color y esperar que no ocurriese ningún incidente. Como efecto secundario de esto, el **+2A** normalmente no puede enviar *secuencias de escape* a la impresora. Por ejemplo, supongamos que la impresora necesita el código de escape (carácter 27) seguido de **"x";CHR$(1)** para activar el modo de alta calidad (NLQ). Con otro ordenador usaríamos una orden del siguiente estilo:

```
LPRINT CHR$(27);"x";CHR$(1);
"Esto es alta calidad"
```

Sin embargo, en el **+2A** necesitamos dar antes la orden:

```
FORMAT LPRINT "U"
```

para pedir a +3 BASIC que no interprete los caracteres como 'códigos de Spectrum', sino como caracteres normales. Sin esta orden, todos los códigos del 165 en adelante (v. Sección 28) serán tratados como 'códigos de Spectrum' y traducidos a palabras clave; además, casi todos los códigos inferiores al 32 serán filtrados.

Si hemos dado la orden anterior, antes de usar **LLIST** tenemos que restablecer la situación normal con:

```
FORMAT LPRINT "E"
```

En resumen:

- Si queremos poder enviar a la impresora el juego de caracteres completo, damos la orden:

  FORMAT LPRINT "U"

- Si después nos ponemos a modificar o escribir un programa y queremos listarlo en la impresora, damos la orden:

  FORMAT LPRINT "E"

La tercera instrucción de +3 BASIC para el control de la impresora, **COPY**, imprime una copia de la pantalla. Para ver una primera demostración, entre en la pantalla pequeña (con la opción **Pantalla** del menú de edición), dé la orden **LIST** para tener en la pantalla algo que imprimir y después escriba:

COPY

La orden **COPY** tarda de 15 a 30 segundos en prepararse para enviar datos a la impresora, así que no se inquiete si no parece que ocurra nada inmediatamente. Al final obtendrá el 'volcado' de la pantalla en la impresora. (Si, en cambio, todo lo que produce **COPY** en la impresora es un montón de caracteres aleatorios, es probable que la impresora no sea del todo compatible.)

En cualquier momento se puede detener la impresión pulsando la tecla [BREAK]. Algunas impresoras tienen lo que se conoce por el nombre de *tampón*, una memoria en la que almacenan el texto antes de imprimirlo. Si su impresora tiene tampón, al pulsar [BREAK] no se detendrá inmediatamente, a pesar de que el **+2A** dejará de enviarle datos en ese mismo momento.

Tenga en cuenta que si interrumpe la orden **COPY** pulsando [BREAK], la impresora puede quedar en modo gráfico, y entonces las siguientes órdenes **LPRINT** producirán una masa de puntos sin sentido o escribirán cada línea parcialmente superpuesta a la anterior. La forma más rápida de volver a la normalidad en este caso es apagar la impresora y volver a encenderla.

Aparte de la simple orden **COPY**, que se limita a dibujar un punto negro en la impresora por cada punto de la pantalla, cualquiera que sea el color de éste, disponemos de una versión ampliada (**COPY EXP**) que imprime diferentes combinaciones de puntos dependiendo de los colores que encuentra en la pantalla. Para probarla, transcriba el siguiente programa:

```
10 FOR b=0 TO 1
20 BRIGHT b
30 FOR i=0 TO 6
40 FOR c=0 TO 31
50 PRINT INK i; i;
60 NEXT c
70 NEXT i
80 NEXT b
```

Seleccione la pantalla inferior (con la opción **Pantalla** del menú de edición) y ejecute el programa. Después dé la orden:

```
COPY EXP
```

El volcado producido por **COPY EXP** es algo más grande que el producido por **COPY**. Como puede apreciar, **COPY EXP** convierte los colores de la pantalla en diferentes densidades de puntos negros. Las zonas en las que habíamos escrito con **BRIGHT 1** quedan en el papel más claras que las correspondientes a **BRIGHT 0**.

La desventaja de **COPY EXP** es su lentitud, pues tarda unos 10 minutos en imprimir el volcado. Sin embargo, es idónea para la reproducción de pantallas gráficas. **COPY** es más adecuada cuando en la pantalla sólo hay texto.

Si la pantalla contiene grandes áreas negras u oscuras, **COPY EXP** gastará excesivamente la cinta de la impresora y tardará bastante más en realizar el volcado que si la pantalla fuera predominantemente blanca. En tales casos se debe incluir la cláusula **INVERSE**:

```
COPY EXP INVERSE
```

cuyo efecto es invertir los puntos en el papel (como en un negativo fotográfico), de modo que las zonas más oscuras de la pantalla resultan más claras en el papel, y viceversa.

INVERSE no puede ser aplicada a **COPY**, sino sólo a **COPY EXP**.

Los volcados producidos por **COPY EXP** y **COPY EXP INVERSE** caben en una hoja de tamaño A4. Sin embargo, hay impresoras que no pueden imprimir a menos de una pulgada (2,5 cm) del borde del papel; en tal caso se puede reducir ligeramente el tamaño del volcado dando la orden:

```
POKE 23419,8
```

El efecto de esta orden es establecer en 8/216" el avance del papel al final de cada pasada de la cabeza impresora. (El **+2A** selecciona el valor 9/126" en el momento de encenderlo.) Una vez seleccionado este nuevo valor, permanece inalterado hasta que demos la orden **NEW**. Al reducir el avance del papel, cada línea se superpone parcialmente a la anterior; de esta forma reducimos el tamaño del volcado, a costa de una pequeña pérdida de calidad.

Si intentamos ejecutar una orden **LLIST**, **LPRINT**, **COPY** o **COPY EXP** cuando la impresora no está conectada (o no está 'en línea'), el **+2A** se pondrá a esperar pacientemente que la impresora (inexistente) le diga que está preparada. Para volver a despertar al ordenador debemos pulsar **BREAK**.

Pruebe este programa:

```
10 FOR n=31 TO 0 STEP -1
20 PRINT AT 31-n,n; CHR$ ( CODE "0"+n);
30 NEXT n
```

Verá una serie de caracteres bajando diagonalmente desde el extremo superior derecho hasta el borde inferior de la pantalla, y en ese momento el programa preguntará si quiere desplazar la pantalla.

Ahora cambie el **AT 31−n,n** de la línea 20 por **TAB n**. El programa producirá exactamente el mismo efecto que antes.

Luego cambie **PRINT** en la línea 20 por **LPRINT**. Esta vez el ordenador no se para a preguntar '**¿MAS?**', pues esto sólo ocurre cuando la salida está siendo dirigida a la pantalla.

Finalmente vuelva a sustituir **TAB n** por **AT 31−n,n** manteniendo **LPRINT**. Esta vez obtendrá solamente una única línea de símbolos. La causa de esta diferencia es que el resultado de **LPRINT** no es impreso inmediatamente, sino que BASIC lo almacena en un tampón hasta que se haya acumulado el equivalente a una línea de impresora, o bien hasta que alguna otra cosa provoque el vaciado del tampón. Así pues, la impresión sólo tiene lugar:

1. Cuando se llena el tampón.
2. Tras una sentencia **LPRINT** que **no** acabe en coma ni en punto y coma.
3. Cuando una coma, apóstrofo o cláusula **TAB** obligue a cambiar de línea.
4. Al final de un programa, si ha quedado algo sin imprimir.
5. En algunas impresoras, cuando se pulsa el botón de 'fuera de línea'.

El punto 3 explica por qué nuestro programa con **TAB** funciona así. En cuanto a **AT**, el número de fila es ignorado y la posición de **LPRINT** (igual que la de **PRINT**) se desplaza a la columna especificada. Una cláusula **AT** no puede hacer nunca que la línea sea enviada a la impresora.

Ejercicios

1. Imprima un gráfico de la función seno: ejecute el primer programa de la Sección 17 de este capítulo y luego dé la orden **COPY**.

2. Ejecute el programa del principio de la Sección 16 y pruebe **COPY EXP** y **COPY EXP INVERSE**.

Sección 22
Canales

Temas tratados:

Dispositivos
Canales
FORMAT, OPEN, CLOSE

El **+2A** puede «leer» información en el teclado, mediante **INPUT** e **INKEY$**, y escribirla en la pantalla o en la impresora mediante **PRINT** y **LPRINT**. Sin embargo, estas órdenes son solamente una parte de un sistema de gestión de entradas y salidas bastante más complejo.

Así, para la orden **PRINT** de BASIC la pantalla y la impresora no son diferentes. En realidad, **PRINT "Rosita"** significa 'toma los caracteres que forman esta palabra y envíalos a cierto sitio', y BASIC sabe que ese 'cierto sitio' suele ser la pantalla. De forma análoga, **LPRINT** normalmente envía los datos a la impresora. Lo que hacen ambas órdenes es enviar los datos a un *dispositivo*.

Los tres dispositivos a los que puede acceder BASIC inicialmente son:

- la pantalla (dispositivo **S**)
- el teclado (dispositivo **K**)
- la impresora (dispositivo **P**)

De ellos, la pantalla es un dispositivo de salida, el teclado es un dispositivo de entrada/salida, la pantalla es un dispositivo de salida si está conectada en la puerta paralelo, y de entrada/salida si está conectada en la puerta serie (**RS232**). Aunque enviar datos al teclado pueda parecer absurdo, lo que hace el ordenador es escribir los caracteres en la pantalla inferior (por ejemplo, cuando ejecutamos una orden **INPUT**).

Para acceder a un dispositivo, antes tenemos que abrirlo, y para abrirlo debemos conectarlo a un *canal*. En BASIC utilizaríamos una orden del tipo:

```
OPEN #4,"p"
```

que significa 'conecta el canal número 4 al dispositivo impresora'. Los canales nos permiten dirigirnos a los dispositivos identificándolos mediante números. De esta forma, podemos diseñar programas que sean capaces de enviar datos a los distintos dispositivos sin tener que usar órdenes diferentes.

PRINT y **LPRINT** son, desde este punto de vista, la misma orden. BASIC normalmente tiene abiertos tres canales:

- el canal **#1** está conectado con el teclado (dispositivo **K**) y es el utilizado por **INPUT** e **INKEY$**;
- el canal **#2** está conectado con la pantalla (dispositivo **S**) y es el utilizado por **PRINT** y **LIST**;
- el canal **#3** está conectado con la impresora (dispositivo **P**) y es el utilizado por **LPRINT** y **LLIST**.

Todas estas órdenes pueden ser dirigidas a cualquier dispositivo incluyendo el signo # seguido del número de canal. Por ejemplo,

```
PRINT #1;"Esta es la pantalla inferior."
```

escribe en la pantalla inferior.

Análogamente,

```
PRINT #3;"No te necesito, LPRINT."
```

se dirige a la impresora.

A la inversa, **LPRINT** puede comportarse como **PRINT**:

```
LPRINT #2;"No me busques en la impresora"
```

donde **LPRINT #2** equivale a **PRINT**.

El siguiente programa ilustra la utilidad de este sistema:

```
10 REM cuadrados para la impresora
20 INPUT "¿Quiere imprimir los resultados (S/N)?";a$
30 LET canal=2
40 IF a$="s" OR a$="S" THEN LET canal=3
50 FOR n=0 TO 10
60 PRINT #canal;n,n*n
70 NEXT n
```

El **+2A** puede manejar hasta 16 canales. BASIC utiliza tres, y otro está reservado para uso interno del ordenador; así pues, quedan 12 disponibles. Por ejemplo,

```
10 REM programa que lee datos en el RS232
20 FORMAT LINE 9600
30 FORMAT LPRINT "r"
40 OPEN #4,"p"
50 PRINT INKEY$ #4;
60 GO TO 50
```

capta los caracteres recibidos en el interfaz serie y los escribe en la pantalla.

Para almacenar directamente en la memoria los caracteres recibidos podemos reemplazar la línea 50 por:

```
50 POKE dirección, CODE (INKEY$ #4)
```

Como ya hemos dicho, la pantalla y la puerta **IMPRESORA** del **+2A** sólo pueden ser utilizadas como dispositivos de salida. Por ejemplo, si intentamos **PRINT INKEY$ #2** provocaremos el mensaje de error '**DISP. E/S INCORRECTO**'.

En teoría es posible desviar los canales de salida normal de BASIC. Por ejemplo, con

```
10 CLOSE #2
20 OPEN #2,"p"
```

desviamos hacia la impresora toda la salida de **PRINT** (que normalmente iría a la pantalla). (La forma en que esto puede afectar al editor de BASIC es impredecible, así que más vale no probarlo.)

En el **+2A** estándar los canales y los dispositivos no tienen demasiado interés. Sin embargo, ciertos periféricos y ampliaciones de BASIC utilizan el sistema de canales para poder realizar funciones más complejas.

Sección 23
IN y OUT

Temas tratados:

IN
OUT

Como el lector ya sabe, podemos leer la memoria ROM y RAM y escribir en la RAM utilizando la función **PEEK** y la orden **POKE**. En realidad, al microprocesador no le importa si la memoria es ROM o RAM: sólo sabe que hay 65536 direcciones de memoria y que puede leer un byte en cada una de ellas (aunque no tenga sentido) y escribir también un byte en cada una (aunque se pierda). Análogamente, hay 65536 *puertas E/S* (puertas de entrada/salida). Estas puertas las utiliza el microprocesador para comunicarse con, por ejemplo, el teclado y la impresora, y también para controlar la memoria adicional y el circuito de sonido. Algunas de ellas pueden ser controladas desde BASIC sin problemas mediante la función **IN** y la orden **OUT**, pero hay posiciones en las que no debemos escribir desde BASIC, ya que seguramente provocaríamos la *caída del sistema*, lo que representaría la pérdida del programa y todos los datos.

IN es una función, como **PEEK**. Su forma es:

IN *dirección*

Tiene un argumento (la dirección de la puerta) y su resultado es el byte leído en esa puerta.

OUT es una sentencia, como **POKE**. Su forma es:

OUT *dirección,valor*

que escribe el *valor* dado en la puerta cuya *dirección* es la especificada. La interpretación que se dé a la dirección depende en gran medida del resto del ordenador. Frecuentemente muchas direcciones diferentes significan lo mismo. En el **+2A** lo más sensato es imaginar la dirección escrita en binario, ya que los bits individuales (cada uno de los cuales puede tener el valor 0 o 1) funcionan independientemente. Una dirección está formada por 16 bits:

A15, A14, A13, A12, A11, A10, A9, A8, A7, A6, A5, A4, A3, A2, A1, A0

A0 es el bit de las unidades, A1 el bit de los 'doses', A2 el bit de los 'cuatros', y así sucesivamente. Los bits A0, A1, A2, A3 y A4 son los importantes. Normalmente tienen el valor 1, y el hecho de que uno de ellos esté a 0 le encarga al ordenador algo específico. El ordenador no puede hacer varias cosas al mismo tiempo, así que no debe estar a 0 más que

uno de estos bits. Los bits A6 y A7 son ignorados, de modo que, si es usted un mago de la electrónica, puede usarlos como quiera. Las mejores direcciones son los múltiplos de 32 menos 1, de tal manera que los bits A0 a A4 estén todos a 1. Los bits A8, A9 y siguientes sirven a veces para dar información adicional; se los utiliza casi siempre para controlar la memoria adicional y el sonido.

El byte leído o escrito (byte de datos) está formado por ocho bits:

D7, D6, D5, D4, D3, D2, D1, D0

Veamos una lista de las direcciones de puerta que pueden ser utilizadas:

Las siguientes direcciones de entrada leen el teclado y el interfaz del magnetófono. El teclado se compone de ocho medias filas de cinco teclas, a saber:

IN 65278 (FFFEh) lee desde [MAYUSCULAS] hasta [V]
IN 65022 (FDFEh) lee desde [A] hasta [G]
IN 64510 (FBFEh) lee desde [Q] hasta [T]
IN 63486 (F7FEh) lee desde [1] hasta [5] (y **JOYSTICK 2**)
IN 61438 (EFFEh) lee desde [0] hasta [6] (y **JOYSTICK 1**)
IN 57342 (DFFEh) lee desde [P] hasta [Y]
IN 49150 (BFFEh) lee desde [INTRO] hasta [H]
IN 32766 (7FFEh) lee desde 'espacio' hasta [B]

[Estas direcciones son 254+256*(255−2↑*n*), donde *n* va de 0 a 7.]

Recuerde que la 'h' significa 'hexadecimal'. Si no conoce usted los números hexadecimales, consulte la Sección 32.

En el byte leído, los bits D0 a D4 representan las cinco teclas de la media fila en cuestión. D0 representa la tecla más externa; D4, la más próxima al centro. El bit es 0 si está pulsada la tecla, y 1 en caso contrario. D6 es controlado por el interfaz del magnetófono; en ausencia de datos procedentes de la cinta, su valor es aleatorio.

Para **JOYSTICK 1**, el bit 0 representa disparo; el bit 1, arriba; el bit 2, abajo; el bit 3, derecha; y el bit 4, izquierda. Para **JOYSTICK 2**, el bit 0 representa izquierda; el bit 1, derecha; el bit 2, abajo; el bit 3, arriba; y el bit 4, disparo. En BASIC éstos se leen como las teclas de números.

La puerta de salida 00FEh (254) controla el sonido (D4) y la señal de grabación hacia el magnetófono (D3) y establece el color del borde (D2, D1 y D0).

Las puertas 00FEh (254), 00F7h (247) y 00EFh (239) están reservadas.

La puerta 7FFDh (32765) gestiona la memoria adicional. La ejecución de **OUT** desde BASIC hacia esa puerta causará casi siempre la caída del sistema, con la consiguiente pérdida del programa y los datos. Esta puerta está descrita en la Sección 24 de este capítulo (en el apartado 'Gestión de la memoria'). Es una puerta de *sólo escritura*, lo que implica que no se puede determinar el estado actual de la paginación mediante una función **IN**. Ésta

es la razón por la que la variable de sistema BANKM está siempre actualizada con el último valor enviado a esta puerta.

La puerta BFFDh (49149) controla los registros de datos del circuito de sonido. La puerta FFFDh (65533) en modo de salida escribe una dirección de registro, y en modo de entrada lee un registro. El uso cuidadoso de estos dos registros puede permitir la generación de sonidos mientras BASIC realiza otra tarea; pero hay que tener en cuenta que también controlan los interfaces **RS232/MIDI** y **AUX**.

La puerta 0FFDh (4093) está relacionada con el interfaz paralelo Centronics (zócalo **IMPRESORA**). Al leerla con **IN**, el bit 0 muestra el estado de la señal BUSY enviada por la impresora. Si ésta se encuentra 'fuera de línea' (o desconectada), ese bit vale 1. Al escribir con **OUT**, esta puerta actúa como registro de salida de datos. Para imprimir un carácter es necesario esperar hasta que BUSY sea 0, escribir el código del carácter en la puerta 0FFDh (4093) y, finalmente, poner el bit de STROBE en la puerta 1FFDh (8189) a nivel bajo y luego otra vez a nivel alto.

La puerta 1FFDh (8189) controla diversos aspectos del **+2A**. Entre otras cosas, decide qué ROM es proyectada hacia el área 0000h...3FFFh (0...16383) de la memoria. Puesto que la puerta es de sólo lectura, +3 BASIC mantiene una variable, BANK678, cuyo contenido es el último valor enviado a esta puerta. Por consiguiente, no se debe escribir valores directamente en esta puerta sin antes comprobar su estado para modificar solamente los bits que interesen. [Lo mismo se puede decir de la puerta 7FFDh (cuyo estado actual se mantiene en BANKM).] Los tres bits menos significativos (0...2) de la puerta 1FFDh sirven para conmutar la RAM y la ROM (véase el apartado 'Gestión de la memoria' en la Sección 24). El bit 3 controla el motor de la unidad de disco (a 0, motor parado; a 1, motor en marcha); no hay ninguna necesidad de escribir en esta puerta para controlar el motor, ya que el sistema operativo +3DOS ofrece rutinas con las que se consigue el mismo efecto. El bit 4 es la señal STROBE de la puerta paralelo, que está activa a nivel bajo; esto quiere decir que, para imprimir el carácter que ha sido enviado a la puerta 0FFDh, primero hay que poner el bit de STROBE a nivel bajo y después volver a ponerlo en su estado normal, que es alto.

Si está conectada la unidad de disco externa (con su interfaz), se puede utilizar las puertas 2FFDh (12285) y 3FFDh (16381) de la siguiente forma:

La puerta 2FFDh (12285) lee el registro de estado principal del circuito controlador del disco (μPD765A). No será útil más que para quienes conozcan a fondo el funcionamiento de este circuito.

La puerta 3FFDh (16381) es el registro de datos del controlador del disco. Se puede leer y escribir en ella, pero no parece probable que tenga ninguna utilidad para los programadores de BASIC. Si se escribe en esta puerta sin saber para qué, es posible que el controlador del disco se confunda, hasta el punto de que las siguientes operaciones de acceso al disco dejen de ser fiables. Además, es muy probable que los experimentos con esta puerta conduzcan a la destrucción de los datos grabados en el disco.

Ejecute este programa para ver cómo funciona el teclado:

```
10 FOR n=0 TO 7: REM numero de media fila
20 LET a=254+256*(255-2↑n)
30 PRINT AT 0,0; IN a: GO TO 30
```

y entreténgase pulsando teclas. Cuando acabe con cada media fila, pulse [BREAK] y escriba:

```
NEXT n
```

Los *buses* de control, datos y direcciones están todos a su disposición en la parte trasera del **+2A**, en el zócalo **EXPANSION E/S**, de modo que usted puede hacer con el **+2A** casi todo lo que hacía con un Z80 (aunque algunas veces el hardware del ordenador puede interferir).

En el Capítulo 10 daremos un diagrama y una descripción de las patillas del zócalo **E/S EXPANSION**.

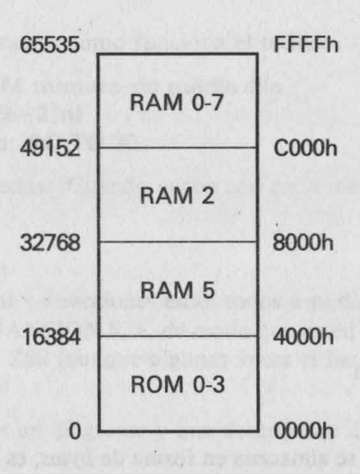
Sección 24
La memoria

Temas tratados:

PEEK
POKE
CLEAR
Gestión de la memoria

En el interior del **+2A** todo se almacena en forma de *bytes*, es decir, números enteros del margen de 0 a 255. Usted puede pensar que ha almacenado el precio de los gambusinos o los nombres de los jugadores del Betis, pero, de hecho, toda la información ha sido convertida en colecciones de bytes, y son los bytes lo que el ordenador ve y entiende.

Cada lugar donde puede ser almacenado un byte está caracterizado por una dirección, que es un número entre 0 (0000h) y 65535 (FFFFh). Esto significa que una dirección puede ser almacenada en forma de dos bytes. Podemos imaginar la memoria como una larga fila de cajas numeradas, cada una de las cuales puede contener un byte. Sin embargo, no todas las cajas son iguales; del 4000h al FFFFh son cajas de RAM, lo que significa que podemos levantar la tapa y alterar el contenido. Pero del 0 al 3FFFh son cajas de ROM, las cuales tienen una tapa de cristal que no podemos abrir; lo único que podemos hacer es ver a través del cristal lo que se puso en ella cuando se fabricó el ordenador. En el **+2A** hemos puesto más del doble de cajas que las que caben cómodamente: mientras que el procesador puede acceder directamente a 65536 bytes, el **+2A** tiene 131072 bytes de RAM y 65536 bytes de ROM, lo que da un total de 196608 bytes (192K). El hardware oculta este exceso de memoria al microprocesador mediante un sistema llamado *paginación*; BASIC y el microprocesador siempre «ven» la memoria como 16K de ROM y 48K de RAM (o 64K de RAM y ninguna ROM, si bien BASIC nunca utiliza esta combinación).



Mapa de memoria del +2A

Para inspeccionar el contenido de una caja usamos la función **PEEK**. Su argumento es la dirección de la caja y su resultado es el contenido. Por ejemplo, el siguiente programa escribe los primeros 21 bytes de la ROM junto con sus direcciones:

```
10 PRINT "Dirección"; TAB 10; "Byte"
20 FOR a=0 TO 20
30 PRINT a; TAB 10; PEEK a
40 NEXT a
```

Estos bytes seguramente no tendrán ningún significado para usted, pero el microprocesador los entiende como instrucciones que le dicen qué tiene que hacer.

Para cambiar el contenido de una caja (suponiendo que sea de RAM), usamos la orden **POKE**. Su forma es:

POKE *dirección,contenido*

donde *dirección* y *contenido* son expresiones numéricas. Por ejemplo, la orden

```
POKE 31000,57
```

da al byte de la dirección 31000 el nuevo valor 57. Escriba:

```
PRINT PEEK 31000
```

para comprobarlo. (Pruebe con otros valores para ver que no estamos haciendo trampa.) El nuevo valor debe estar entre −255 y +255; si es negativo, BASIC le suma 256.

La facultad de modificar el contenido de una posición de memoria nos otorga un inmenso poder sobre el ordenador si sabemos cómo usarla, e inmensas posibilidades de destrucción

si no sabemos. Es muy fácil, colocando un valor incorrecto en ciertas direcciones de memoria, perder largos programas que hemos tardado horas en transcribir. Sin embargo, afortunadamente, esto no le hará al ordenador ningún daño permanente.

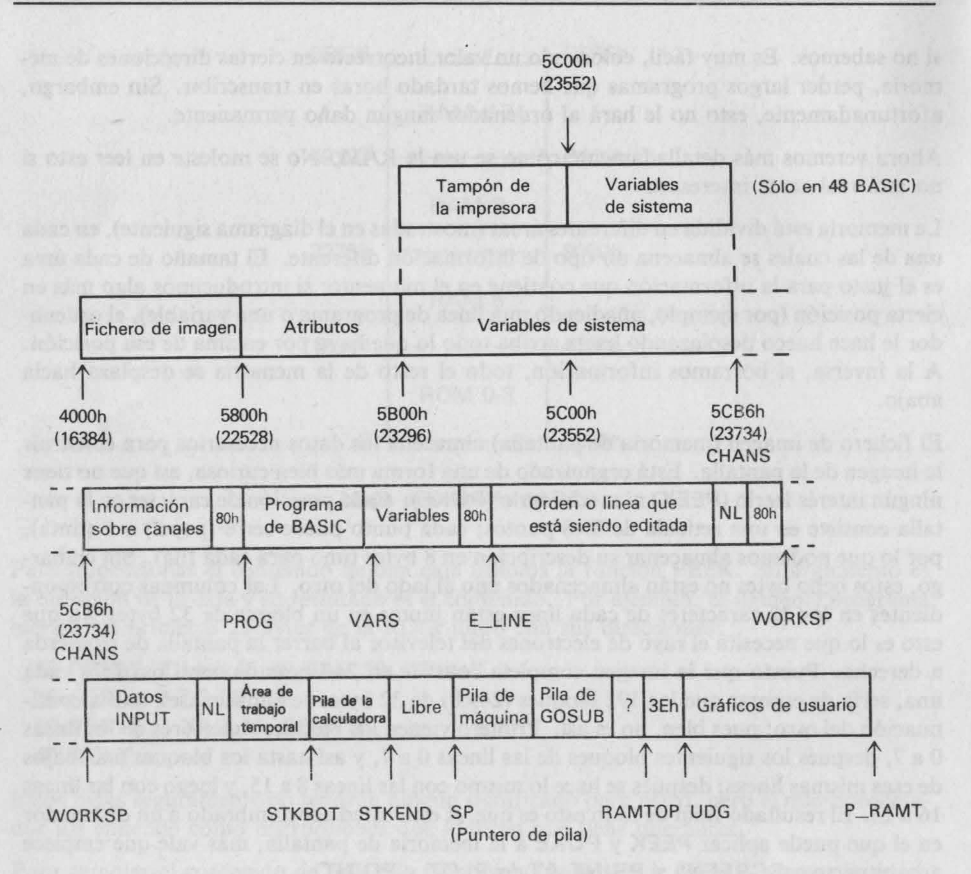
Ahora veremos más detalladamente cómo se usa la RAM. No se moleste en leer esto si no está realmente interesado.

La memoria está dividida en diferentes áreas (mostradas en el diagrama siguiente), en cada una de las cuales se almacena un tipo de información diferente. El tamaño de cada área es el justo para la información que contiene en el momento; si introducimos algo más en cierta posición (por ejemplo, añadiendo una línea de programa o una variable), el ordenador le hace hueco desplazando hacia arriba todo lo que haya por encima de esa posición. A la inversa, si borramos información, todo el resto de la memoria se desplaza hacia abajo.

El fichero de imagen (memoria de pantalla) almacena los datos necesarios para construir la imagen de la pantalla. Está organizado de una forma más bien curiosa, así que no tiene ningún interés leerlo (**PEEK**) ni modificarlo (**POKE**). Cada posición de carácter en la pantalla consiste en una retícula de 8×8 puntos; cada punto puede ser 0 (papel) o 1 (tinta), por lo que podemos almacenar su descripción en 8 bytes (uno para cada fila). Sin embargo, estos ocho bytes no están almacenados uno al lado del otro. Las columnas correspondientes en los 32 caracteres de cada línea están juntas en un bloque de 32 bytes, ya que esto es lo que necesita el rayo de electrones del televisor al barrer la pantalla de izquierda a derecha. Puesto que la imagen completa consiste en 24 líneas de ocho barridos cada una, sería de esperar que los 192 bloques (24×8) de 32 bytes se almacenasen uno a continuación del otro; pues bien, no es así. Primero vienen los bloques superiores de las líneas 0 a 7, después los siguientes bloques de las líneas 0 a 7, y así hasta los bloques más bajos de esas mismas líneas; después se hace lo mismo con las líneas 8 a 15, y luego con las líneas 16 a 23. El resultado final de todo esto es que, si está usted acostumbrado a un ordenador en el que puede aplicar **PEEK** y **POKE** a la memoria de pantalla, más vale que empiece a habituarse a **SCREEN$** y **PRINT AT** (o **PLOT** y **POINT**).

Los *atributos* de cada posición de carácter son los colores y las restantes características; su formato es el que describimos a propósito de la función **ATTR** y sí son almacenados línea por línea en el orden que usted esperaría.

La forma en la que el ordenador organiza sus asuntos no es exactamente la misma en 48 BASIC y en +3 BASIC. El área que constituía el tampón de la impresora en 48 BASIC se utiliza en +3 BASIC para almacenar ciertas variables de sistema adicionales. (La estructura es similar a la del Spectrum **+2**, pero las variables han cambiado.)

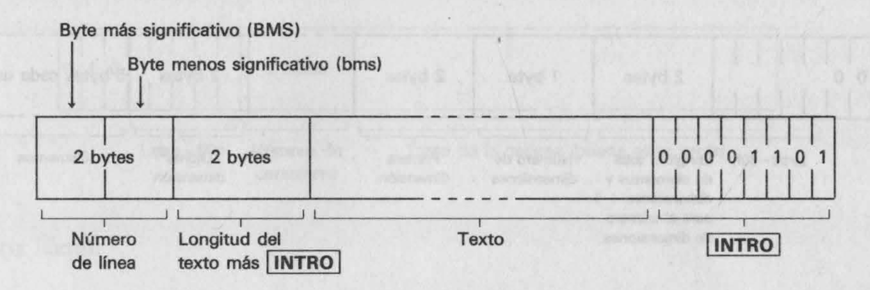


Mapa de la memoria de BASIC

Las variables de sistema contienen diversos elementos de información que indican al ordenador en qué estado se encuentra. La lista completa está en la Sección 26 de este capítulo; por el momento observe que algunas de ellas (CHANS, PROG, VARS, E_LINE, etc.) contienen la dirección de los límites entre las distintas áreas de la memoria. No son variables de BASIC y por lo tanto no serán reconocidas por el **+2A**.

La información sobre canales contiene datos sobre los dispositivos de entrada y salida: el teclado (junto con la pantalla inferior), la pantalla superior y la impresora.

Cada línea de programa de BASIC tiene la siguiente forma:

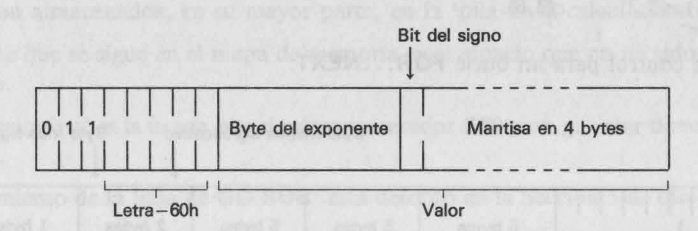


Observe que, en contraste con todos los demás casos de números de dos bytes en el Z80, aquí el número de línea se almacena con su byte más significativo primero, es decir, en el mismo orden en que nosotros los escribimos.

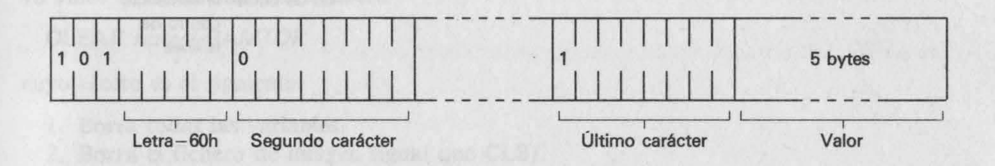
Una constante numérica del programa lleva a continuación su forma binaria, usando el carácter **CHR$ 14** seguido por cinco bytes para el número propiamente dicho.

Las variables tienen formatos diferentes para los distintos tipos. Las letras de los nombres se almacenan como si empezasen en minúscula.

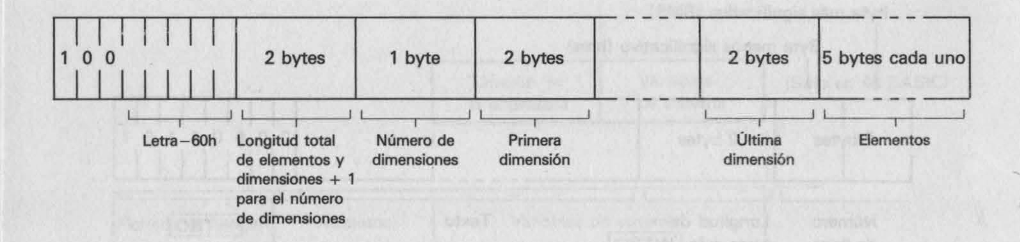
Variable numérica cuyo nombre es una sola letra:



Variable numérica cuyo nombre consiste en varias letras:



Matriz numérica:



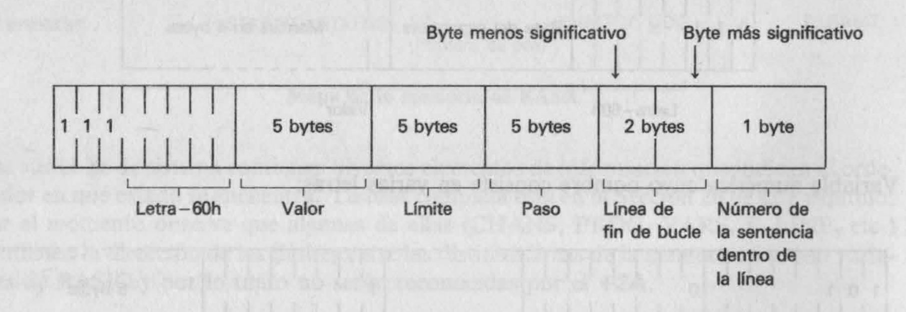
El orden de los elementos es:

en primer lugar los elementos cuyo primer subíndice es 1
a continuación los elementos cuyo primer subíndice es 2
a continuación los elementos cuyo primer subíndice es 3
. . . y así sucesivamente para todos los posibles valores del primer subíndice.

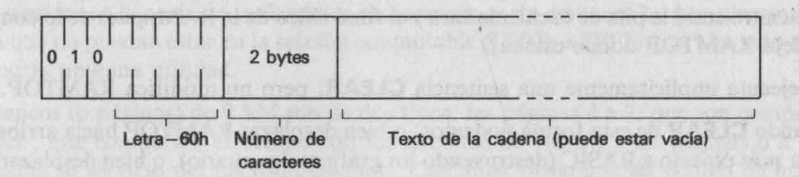
Los elementos con un primer subíndice dado están ordenados de la misma forma con respecto al segundo, y así sucesivamente hasta el último.

Por ejemplo, los elementos de la matriz **c** de 3*6 de la Sección 12 de este capítulo son almacenados en el siguiente orden: *c*(1,1) *c*(1,2) *c*(1,3) *c*(1,4) *c*(1,5) *c*(1,6) *c*(2,1) *c*(2,2) . . . *c*(2,6) *c*(3,1) *c*(3,2) . . . *c*(3,6).

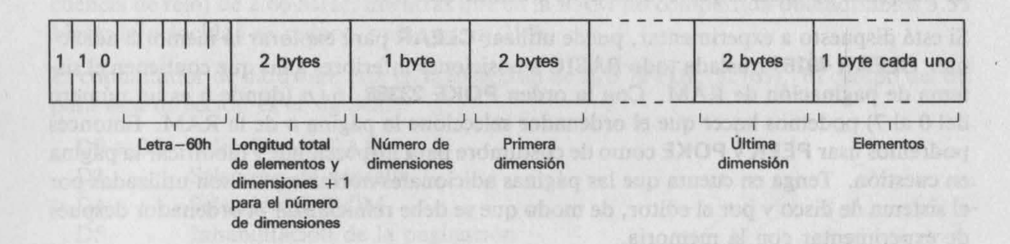
Variable de control para un bucle **FOR. . .NEXT**:



Variable literal:



Matriz literal:



La calculadora es la parte de BASIC que se encarga de la aritmética; los números con los que opera son almacenados, en su mayor parte, en la 'pila de la calculadora'.

La parte libre que se sigue en el mapa de memoria es el espacio que no ha sido utilizado de momento.

La 'pila de máquina' es la usada por el microprocesador Z80 para guardar direcciones de retorno, etc.

El funcionamiento de la 'pila de **GO SUB**' está descrito en la Sección 5 de este capítulo.

El byte al que «apunta» RAMTOP tiene la dirección más alta utilizada por BASIC. Incluso **NEW**, que borra la RAM, no pasa de esa dirección, y por tanto no cambia los gráficos definidos por el usuario. Se puede modificar la dirección RAMTOP especificando su nuevo valor en una sentencia **CLEAR**:

CLEAR *nueva-RAMTOP*

cuyo efecto es el siguiente:

1. Borra todas las variables.
2. Borra el fichero de imagen (igual que **CLS**).
3. Restablece la posición del cursor gráfico en el extremo inferior izquierdo de la pantalla.

4. Restaura (**RESTORE**) el puntero de datos.
5. Borra la pila de **GO SUB** e iguala su base a RAMTOP (suponiendo que ésta se encuentre entre la pila de la calculadora y el final físico de la RAM, pues de lo contrario deja RAMTOP donde estaba).

RUN ejecuta implícitamente una sentencia **CLEAR**, pero no modifica RAMTOP.

Utilizando **CLEAR** de esta forma podemos, o bien desplazar RAMTOP hacia arriba para dedicar más espacio a BASIC (destruyendo los gráficos de usuario), o bien desplazarla hacia abajo para hacer más grande la zona de RAM que no es borrada por **NEW**. También la podemos usar para trasladar la pila de máquina a alguna posición por debajo de BFE0h (49120) cuando vamos a hacer llamadas a +3DOS; de esta forma el programa de código de máquina no tendrá que ocuparse de trasladarla.

Si está dispuesto a experimentar, puede utilizar **CLEAR** para explorar la memoria adicional. **CLEAR 49151** traslada todo BASIC a posiciones inferiores a las que contienen el sistema de paginación de RAM. Con la orden **POKE 23388**,16+*n* (donde *n* es un número del 0 al 7) podemos hacer que el ordenador seleccione la página *n* de la RAM. Entonces podremos usar **PEEK** y **POKE** como de costumbre para inspeccionar y modificar la página en cuestión. Tenga en cuenta que las páginas adicionales normalmente son utilizadas por el sistema de disco y por el editor, de modo que se debe reinicializar el ordenador después de experimentar con la memoria.

Escriba **NEW** y luego **CLEAR 23825** para hacerse una idea de lo que le ocurre a la máquina cuando se queda sin memoria.

Si intenta que el **+2A** calcule algo (escriba, por ejemplo, **PRINT 1+1**), verá en la pantalla el mensaje **'4 MEMORIA AGOTADA'**. Esto significa que al ordenador no le queda más espacio donde almacenar información. Si se le presenta este mensaje cuando está transcribiendo un programa largo, tendrá que vaciar un poco la memoria (por ejemplo, borrando una línea) para poder controlar el ordenador.

Gestión de la memoria

Ya hemos mencionado antes que en el ordenador hay bastante más memoria que la que el microprocesador puede manejar cómodamente. Éste puede «direccionar» sólo 64K de memoria de una vez, pero la memoria adicional puede ser intercambiada a discreción con esos 64K. Observe un aparato de televisión; a pesar de que sólo podemos ocuparnos de un canal a la vez, hay otros canales que pueden ser seleccionados pulsando los botones adecuados. Así, aunque hay más información de la que somos capaces de asimilar a un tiempo, podemos elegir la que nos interese.

Lo que ocurre en el ordenador es muy parecido. Colocando los bits adecuados en cierta puerta de E/S, la máquina puede elegir qué porciones quiere usar del total de 192K de memoria. BASIC ignora casi permanentemente toda la memoria adicional. En cambio, en

los programas de juegos viene muy bien tener una RAM triplicada. Vuelva a echar un vistazo al mapa de memoria del principio de esta sección. Las páginas 2 y 5 de la RAM están siempre en la posición indicada en el diagrama, y sin embargo no hay ninguna razón por la que no puedan estar en la sección conmutable (C000h a FFFFh), si bien esto tampoco tendría ninguna utilidad.

Los bancos (o páginas) de RAM son de dos tipos: las páginas 4 a 7, que son compartidas (es decir, que comparten el tiempo con los circuitos de video), y las páginas 0 a 3, que sólo están al servicio del microprocesador. Cualquier programa de código de máquina que requiera un control muy preciso del tiempo (por ejemplo, los de música o comunicaciones) debe guardar todas las rutinas de temporización en los bancos no compartidos. Por ejemplo, ejecutando instrucciones NOP en la RAM compartida obtenemos una frecuencia de reloj de 2.66 MHz, mientras que en la RAM no compartida obtendríamos 3.55 MHz; la velocidad se reduce, pues, en un 25%.

El conmutador de hardware está en la dirección de E/S 7FFDh (32765). El campo del bit para esta dirección es el siguiente:

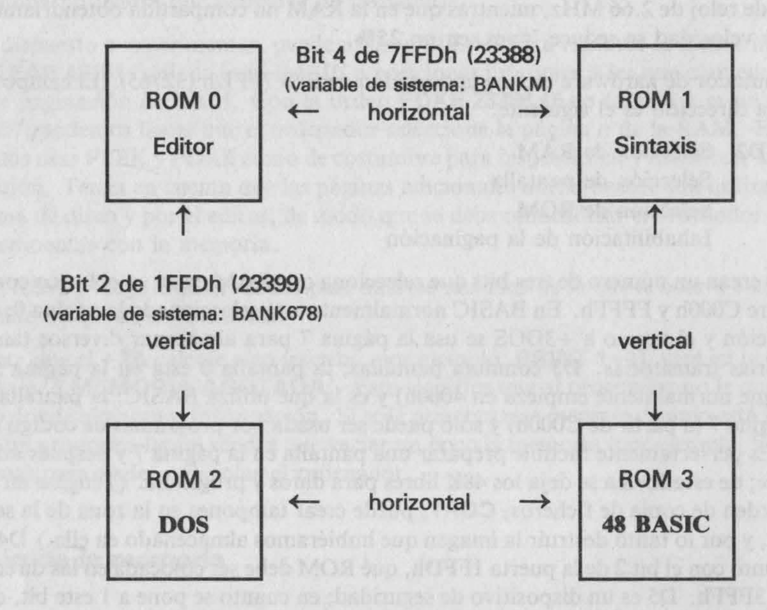
D0 a D2	Selección de RAM
D3	Selección de pantalla
D4	Selección de ROM
D5	Inhabilitación de la paginación

D2 a D0 crean un número de tres bits que selecciona qué RAM entra en el hueco comprendido entre C000h y FFFFh. En BASIC normalmente está seleccionada la página 0; durante la edición y el acceso a +3DOS se usa la página 7 para almacenar diversos tampones y memorias transitorias. D3 conmuta pantallas; la pantalla 0 está en la página 5 de la RAM (que normalmente empieza en 4000h) y es la que utiliza BASIC; la pantalla 1 está en la página 7 (a partir de C000h) y sólo puede ser usada por programas de código de máquina. Es perfectamente factible preparar una pantalla en la página 7 y después «desconmutarla»; de esta forma se deja los 48K libres para datos y programa. (Téngase en cuenta que la orden de copia de ficheros, **COPY**, puede crear tampones en la zona de la segunda pantalla, y por lo tanto destruir la imagen que hubiéramos almacenado en ella.) D4 determina, junto con el bit 2 de la puerta 1FFDh, qué ROM debe ser colocada en las direcciones 0000h a 3FFFh. D5 es un dispositivo de seguridad; en cuanto se pone a 1 este bit, quedan imposibilitadas todas las operaciones de paginación. Este sistema es el utilizado cuando el ordenador adopta la configuración del Spectrum de 48K estándar y los circuitos de paginación de memoria están bloqueados. A partir de ese momento no se lo puede volver a convertir en un ordenador de 128K más que apagándolo o pulsando el botón **REINIC**; no obstante, el circuito de sonido puede seguir siendo controlado por **OUT**. Si se carga desde el disco un programa de juego diseñado para el Spectrum de 48K y no funciona, quizá lo haga si se da la orden **SPECTRUM** y luego **OUT 32765,48** (que bloquea el bit 5 en esta puerta).

El **+2A** utiliza la puerta de E/S 1FFDh para realizar ciertas operaciones de conmutación de ROM y RAM. El campo del bit para esta dirección es el siguiente:

D0 y D1	Conmutación de ROM/RAM
D3	Decide si D0 y D1 afectan a la ROM o a la RAM
D4	Motor del disco
D5	Señal STROBE en la puerta paralelo (activa a nivel bajo)

Cuando el bit 0 está a 0, el bit 1 es indiferente y el bit 2 constituye un conmutador 'vertical' para la ROM (que elige entre ROM 0 y ROM 2, o entre ROM 1 y ROM 3). El bit 4 de la puerta 7FFDh es un conmutador 'horizontal' para la ROM (que elige entre ROM 0 y ROM 1, o entre ROM 2 y ROM 3). El siguiente diagrama ilustra las diferentes posibilidades:



Conmutación horizontal y vertical de la ROM

Es útil imaginar que el bit 4 de la puerta 7FFDh y el bit 2 de la puerta 1FFDh se combinan para formar un número de dos bits (margen de 0 a 3) que determina qué ROM entra en la zona de 0000h a 3FFFh. Al formar ese número, el bit 4 de la puerta 7FFDh sería el menos significativo, y el otro el más significativo.

Bit 2 de 1FFDh (Variable de sistema: BANK678)	Bit 4 de 7FFDh (Variable de sistema: BANKM)	ROM que entra en 0000h-3FFFh
0	0	0
0	1	1
1	0	2
1	1	3

Conmutación de ROM (con el bit 0 de 1FFDh puesto a 0)

Cuando el bit 0 de la puerta 1FFDh está a 1, los bits 1 y 2 controlan qué combinación de páginas de RAM ocupan los 64K posibles. Estas combinaciones no son utilizadas por BASIC, pero pueden serlo por los autores de sistemas operativos y programas de juegos. Cuando se invoca la rutina 'DOS CARGAR', el cargador inicial es transferido al «entorno» de las páginas 4, 7, 6, 3. Las opciones de paginación de RAM del **+2A** son:

Bit 2 de 1FFDh	Bit 1 de 1FFDh	Páginas de RAM que entran en 0000h-3FFFh, 4000h-4FFFh, etc.
0	0	0, 1, 2, 3
0	1	4, 5, 6, 7
1	0	4, 5, 6, 3
1	1	4, 7, 6, 3

Conmutación de páginas de RAM (con el bit 0 de 1FFDh puesto a 1)

Sección 25
Variables de sistema

Temas tratados:

POKE, PEEK

Los bytes de memoria del 5B00h (23296) al 5CB6h (23734) están reservados para usos específicos del sistema. Hay también unas cuantas rutinas (utilizadas para mantener en orden la paginación de memoria) y algunas posiciones que contienen *variables de sistema*. Éstas pueden ser examinadas (con **PEEK**) para conocer diversos aspectos del sistema; también puede tener interés modificar algunas de ellas. En esta sección daremos la lista completa.

Hay gran diferencia, como sería de esperar, entre el área de variables de sistema de 48 BASIC y la de +3 BASIC. En el modo de 48 BASIC las rutinas y variables por debajo de 5C00h (23552) no existen; en su lugar hay un tampón, entre 5B00h (23296) y 5C00h (23552), que se usa para controlar la impresora. Ésta era una posición muy popular para pequeñas rutinas de código de máquina en el Spectrum de 48K; si usted prueba cualquiera de estas rutinas en +3 BASIC, provocará la caída del sistema con toda seguridad. Todos los programas antiguos que usan **PEEK**, **POKE** y **USR** deben ser ejecutados, por tanto, en 48 BASIC. (No obstante, se los puede introducir en +3 BASIC y luego pasar a 48 BASIC con la orden **SPECTRUM**.) Si se prevé la posibilidad de que un programa modifique las puertas de E/S adicionales del **+2A**, se debe dar una orden **OUT 32765,48** para poner a 1 el bit 5 de la puerta 7FFDh, con lo que se hace imposible la conmutación de la ROM y la RAM.

Las variables de sistema tienen nombres, pero es preciso no confundirlos con las palabras clave y los nombres de variable utilizados en BASIC. El ordenador no reconocerá los nombres como referencias a variables de sistema; nosotros los utilizamos solamente como nemónicos y para el ordenador no significan nada.

Las abreviaturas de la columna 1 de la tabla significan lo siguiente:

X — No se debe modificar estas variables porque el sistema podría fallar.

N — La modificación de las variables no tendrá efectos duraderos.

R — No es una variable, sino el punto de entrada a una rutina.

El número de la columna 1 es el número de bytes de que consta la variable o rutina. En el caso de dos bytes, el primero es el menos significativo, lo contrario de lo que parecería natural. Así, para escribir el valor **v** en una variable de dos bytes que se encuentra en la dirección **n** se debe dar las órdenes:

```
POKE n,v–256* INT (v/256)
POKE n+1, INT (v/256)
```

Para leer ese valor se utiliza la expresión:

```
PEEK n+256* PEEK (n+1)
```

Notas	Dirección hex (decimal)	Nombre	Contenido
R16	5B00h (23296)	SWAP	Subrutina de paginación.
R17	5B10h (23312)	STOO	Subrutina de paginación. Se entra en ella con las interrupciones inhibidas y AF y BC en la pila.
R9	5B21h (23329)	YOUNGER	Subrutina de paginación.
R16	5B2Ah (23338)	REGNUOY	Subrutina de paginación.
R24	5B3Ah (23354)	ON_ERR	Subrutina de paginación.
X2	5B52h (23378)	OLDHL	Almacén provisional para registros durante la conmutación de las ROM.
X2	5B54h (23380)	OLDBC	Almacén provisional para registros durante la conmutación de las ROM.
X2	5B56h (23382)	OLDAF	Almacén provisional para registros durante la conmutación de las ROM.
N2	5B58h (23384)	TARGET	Dirección de subrutina en ROM 3.
X2	5B5Ah (23386)	RETADDR	Dirección de retorno en ROM 1.
X1	5B5Ch (23388)	BANKM	Copia del último byte enviado a la puerta de E/S 7FFDh (32765). Esta puerta controla la paginación de RAM (bits 0 a 2), la conmutación 'horizontal' de las ROM (0↔1 y 2↔3, bit 4), la selección de pantallas (bit 3) y la inhabilitación de la paginación (bit 5). Este byte tiene que ser actualizado con el último valor enviado a la puerta si las interrupciones están habilitadas.
X1	5B5Dh (23389)	RAMRST	Instrucción RST8. Utilizada por ROM 1 para informar de errores antiguos a ROM 3.
N1	5B5Eh (23390)	RAMERR	Número de error entregado por ROM 1 a ROM 3. También utilizada por **SAVE** y **LOAD** como almacén temporal.
2	5B5Fh (23391)	BAUD	Periodo de bit del RS232 en estados T/26. Establecido por **FORMAT LINE**.
N2	5B61h (23393)	SERFL	Indicador de recepción del segundo carácter, y datos.
N1	5B63h (23395)	COL	Columna actual, desde la 1 hasta la anchura.
1	5B64h (23396)	WIDTH	Anchura del papel en columnas. (Por defecto, 80.)
1	5B65h (23397)	TVPARS	Número de parámetros por línea esperados por el RS232.

Notas	Dirección hex (decimal)	Nombre	Contenido
1	5B66h (23398)	FLAGS3	Diversos indicadores. Los bits 0, 1 y 6 no son útiles. El bit 2 está a 1 si los códigos de palabra clave deben ser convertidos en texto al imprimir. El bit 3 está a 1 si la salida de impresora se dirige al RS232; por defecto (bit 3 a 0), la salida va a la puerta Centronics. El bit 4 está a 1 si existe un interfaz de disco. El bit 5 está a 1 si está conectada la unidad B.
X1	5B67h (23399)	BANK678	Copia del último byte enviado a la puerta de E/S 1FFDh (8189). Esta puerta controla la paginación de RAM y la conmutación de las ROM (bits 0 a 2; si el bit 0 está a 0, el bit 2 controla la conmutación 'vertical' de las ROM, 0↔2 y 1↔3), el motor del disco (bit 3) si está conectada la unidad externa, y la señal STROBE de la puerta Centronics (bit 4). Este byte tiene que ser actualizado con el último valor enviado a la puerta si las interrupciones están habilitadas.
N1	5B68h (23400)	XLOC	Contiene la posición horizontal cuando se está ejecutando la orden **COPY** no ampliada.
N1	5B69h (23401)	YLOC	Contiene la posición vertical cuando se está ejecutando la orden **COPY** no ampliada.
X2	5B6Ah (23402)	OLDSP	Contiene el valor antiguo de SP (puntero de pila) cuando se está usando TSTACK.
X2	5B6Ch (23404)	SYNRET	Dirección de retorno para ONERR.
5	5B6Eh (23406)	LASTV	Último valor escrito por la calculadora.
2	5B73h (23411)	RCLINE	Línea que está siendo renumerada.
2	5B75h (23413)	RCSTART	Número de la primera línea para la renumeración (por defecto, 10).
2	5B77h (23415)	RCSTEP	Incremento entre líneas para la renumeración (por defecto, 10).
1	5B79h (23417)	LODDRV	Contiene **T** si **LOAD**, **VERIFY** y **MERGE** están desviadas hacia la cinta; de lo contrario, contiene **A**, **B** o **M**.
1	5B7Ah (23418)	SAVDRV	Contiene **T** si **SAVE** está desviada hacia la cinta; de lo contrario, contiene **A**, **B** o **M**.
1	5B7Bh (23419)	DUMPLF	Contiene la interlínea (en unidades de 1/216 pulgadas) para los avances del papel al ejecutar **COPY EXP**. Normalmente es 9, pero se puede escribir 8 en esta dirección si da problemas el hecho de que la impresora detecte el final del papel al realizar volcados de pantalla sobre hojas de tamaño A4. De esta forma se reduce ligeramente el tamaño del volcado y se mejora la relación de aspecto, pero se pierde algo de calidad.
N8	5B7Ch (23420)	STRIP1	Mapa de bits de la banda superior del menú.

Notas	Dirección hex (decimal)	Nombre	Contenido
N8	5B84h (23428)	STRIP2	Mapa de bits de la banda inferior del editor. Llega hasta 5B8Bh (23436).
X115	5BFFh (23551)	TSTACK	La pila temporal crece desde aquí hacia abajo. Se utiliza esta pila cuando la página 7 de RAM está conmutada en la zona superior de la memoria (durante la ejecución del editor y en las llamadas a +3DOS). Puede descender hasta 5B8Ch (y, si es necesario, invadir STRIP1 y STRIP2). De esta forma se dispone de una pila de al menos 115 bytes cuando BASIC invoca rutinas de +3DOS.
N8	5C00h (23552)	KSTATE	Usada en la lectura del teclado.
N1	5C08h (23560)	LASTK	Última tecla pulsada.
1	5C09h (23561)	REPDEL	Tiempo, en cincuentavos de segundo, que se debe mantener pulsada una tecla para que empiece a repetirse. Inicialmente es 35, pero se le puede dar otros valores (con **POKE**).
1	5C0Ah (23562)	REPPER	Pausa, en cincuentavos de segundo, entre las repeticiones sucesivas de una tecla pulsada (inicialmente, 5).
N2	5C0Bh (23563)	DEFADD	Si se está evaluando una función definida por el usuario, dirección de los argumentos; de lo contrario, 0.
N1	5C0Dh (23565)	KDATA	Segundo byte de los controles de color introducidos por el teclado.
N2	5C0Eh (23566)	TVDATA	Almacena bytes de los controles de color, **AT** y **TAB** que van al televisor.
X38	5C10h (23568)	STRMS	Direcciones de los dispositivos conectados a los canales.
2	5C36h (23606)	CHARS	256 menos que la dirección del juego de caracteres (que empieza con el espacio y continúa hasta el símbolo de copyright, ©). Normalmente en ROM, pero se puede elaborar un juego en la RAM y hacer que CHARS apunte hacia él.
1	5C38h (23608)	RASP	Duración del pitido de aviso.
1	5C39h (23609)	PIP	Duración del 'click' del teclado.
1	5C3Ah (23610)	ERR_NR	1 menos que el código del informe. Empieza en 255 (para −1), de forma que **PEEK 23610** da 255.
X1	5C3Bh (23611)	FLAGS	Diversos indicadores que controlan el sistema BASIC.
X1	5C3Ch (23612)	TVFLAG	Indicadores relacionados con el televisor.
X2	5C3Dh (23613)	ERR_SP	Dirección del elemento de la pila de máquina que hay que usar como retorno de error.
N2	5C3Fh (23615)	LIST_SP	Dirección de la dirección de retorno desde un listado automático.
N1	5C41h (23617)	MODE	Especifica cursor **K, L, C, E** o **G**.
2	5C42h (23618)	NEWPPC	Línea a la que hay que saltar.

Notas	Dirección hex (decimal)	Nombre	Contenido
1	5C44h (23620)	NSPPC	Número de sentencia dentro de una línea al que hay que saltar. La modificación de NEWPPC primero y de NSPPC después fuerza el salto a una sentencia específica dentro de una línea.
2	5C45h (23621)	PPC	Número de línea de la sentencia que está siendo ejecutada.
1	5C47h (23623)	SUBPPC	Número, dentro de la línea, de la sentencia que está siendo ejecutada.
1	5C48h (23624)	BORDCR	Color del borde multiplicado por 8; también contiene los atributos usados normalmente para la parte inferior de la pantalla.
2	5C49h (23625)	EPPC	Número de la línea actual (con el cursor de programa).
X2	5C4Bh (23627)	VARS	Dirección de variables.
N2	5C4Dh (23629)	DEST	Dirección de la variable en asignación.
X2	5C4Fh (23631)	CHANS	Dirección de los datos del canal.
X2	5C51h (23633)	CURCHL	Dirección de la información que está siendo usada para entrada y salida.
X2	5C53h (23635)	PROG	Dirección del programa de BASIC.
X2	5C55h (23637)	NXTLIN	Dirección de la siguiente línea del programa.
X2	5C57h (23639)	DATADD	Dirección del terminador del último elemento de **DATA**.
X2	5C59h (23641)	E_LINE	Dirección de la orden que está siendo escrita en el teclado.
2	5C5Bh (23643)	K_CUR	Dirección del cursor.
X2	5C5Dh (23645)	CHADD	Dirección del próximo carácter que debe ser interpretado (el carácter que sigue al argumento de **PEEK**, o el código de 'línea nueva' al final de una sentencia **POKE**.
2	5C5Fh (23647)	X_PTR	Dirección del carácter que está detrás del marcador [?].
X2	5C61h (23649)	WORKSP	Dirección de área de trabajo temporal.
X2	5C63h (23651)	STKBOT	Dirección del extremo inferior de la pila de la calculadora.
X2	5C65h (23653)	STKEND	Dirección del comienzo del espacio libre.
N1	5C67h (23655)	BREG	Registro *b* de la calculadora.
N2	5C68h (23656)	MEM	Dirección del área usada para memoria de la calculadora. (Generalmente MEMBOT, pero no siempre.)
1	5C6Ah (23658)	FLAGS2	Otros indicadores. (El bit 3 se pone a 1 cuando el teclado está en mayúsculas.)
X1	5C6Bh (23659)	DF_SZ	Número de líneas (incluida una en blanco) de que consta la pantalla inferior.
2	5C6Ch (23660)	S_TOP	El número de la primera línea en listados automáticos.
2	5C6Eh (23662)	OLDPPC	Número de la línea a la cual salta **CONTINUE**.
1	5C70h (23664)	OSPCC	Número de la sentencia, dentro de la línea, a la cual salta **CONTINUE**.
N1	5C71h (23665)	FLAGX	Diversos indicadores.
N2	5C72h (23666)	STRLEN	Longitud de la cadena que está siendo asignada.
N2	5C74h (23668)	TADDR	Dirección del siguiente elemento en la tabla de sintaxis (muy improbable que sea útil).

Notas	Dirección hex (decimal)	Nombre	Contenido
2	5C76h (23670)	SEED	Semilla para **RND**. Ésta es la variable establecida por **RANDOMIZE**.
3	5C78h (23672)	FRAMES	Contador de barridos del cuadro en 3 bytes (primero el byte menos significativo); se incrementa cada 20 ms.
2	5C7Bh (23675)	UDG	Dirección del primer gráfico definido por el usuario. Puede interesar cambiarla, por ejemplo, para ahorrar espacio limitando el número de gráficos definibles.
1	5C7Dh (23677)	COORDS	Coordenada *x* del último punto dibujado.
1	5C7Eh (23678)		Coordenada *y* del último punto dibujado.
1	5C7Fh (23679)	P_POSN	33 menos número de columna de la posición de la cabeza impresora.
1	5C80h (23680)	PR_CC	Byte menos significativo de la dirección de la próxima posición en la que debe escribir **LPRINT** (en el tampón de la impresora).
1	5C81h (23681)		No utilizada.
2	5C82h (23682)	ECHO_E	33 menos número de columna y 24 menos número de fila (pantalla inferior) del final del tampón de entrada.
2	5C84h (23684)	DF_CC	Dirección, en el fichero de imagen, de la posición de escritura.
2	5C86h (23686)	DF_CCL	Como DF CC, pero para la pantalla inferior.
X1	5C88h (23688)	S_POSN	33 menos número de columna de la posición de **PRINT**.
X1	5C89h (23689)		24 menos número de fila de la posición de **PRINT**.
X2	5C8Ah (23690)	S_POSNL	Como S_POSN, pero para la pantalla inferior.
1	5C8Ch (23692)	SCR_CT	Cuenta los desplazamientos de la pantalla. Es siempre 1 más que el número de desplazamientos que serán realizados antes de emitir la pregunta '**¿MAS?**'. Si modificamos este número de vez en cuando introduciendo valores mayores que 1 (por ejemplo, 255), la pantalla rodará sin pedir permiso.
1	5C8Dh (23693)	ATTR_P	Colores permanentes actuales, etc. (establecidos por las instrucciones de color).
1	5C8Eh (23694)	MASK_P	Usada para colores transparentes, etc. Cualquier bit que sea 1 indica que el atributo correspondiente no se toma de ATTR_P, sino de lo que ya está en la pantalla.
N1	5C8Fh (23695)	ATTR_T	Colores temporales actuales, etc. (establecidos por cláusulas de color).
N1	5C90h (23696)	MASK_T	Como MASK_P, pero para características temporales.
1	5C91h (23697)	P_FLAG	Otros indicadores.
N30	5C92h (23698)	MEMBOT	Área de memoria de la calculadora, usada para almacenar números que no interesa introducir en la pila de la calculadora.

Notas	Dirección hex (decimal)	Nombre	Contenido
2	5CB0h (23728)	NMIADD	Contiene la dirección de la rutina de servicio de interrupciones no enmascarables (NMI). (*Nota.* En ordenadores anteriores esto no funcionaba correctamente y en la documentación se decía que estos dos bytes estaban 'no utilizados'. Los programas que utilicen estos dos bytes como almacén temporal deben ser modificados.).
2	5CB2h (23730)	RAMTOP	Dirección del último byte del área de sistema de BASIC.
2	5CB4h (23732)	P_RAMT	Dirección del último byte de la RAM física.

Sección 26
Utilización del código de máquina

Temas tratados:

USR con argumento numérico

Esta sección está escrita para quienes ya conocen el *código de máquina* del Z80, es decir, el juego de instrucciones a las que responde el microprocesador Z80. Si usted no sabe nada sobre código de máquina y tiene interés en aprenderlo, puede leer alguno de los muchos libros que se ha escrito sobre el tema; debería adquirir uno que se titulase 'Código de máquina (o lenguaje ensamblador) en el Z80 para el principiante absoluto', o algo así; y si en él se menciona el **+2A** o los otros ordenadores ZX Spectrum, tanto mejor.

Normalmente se suele escribir los programas de código de máquina en *lenguaje ensamblador*, el cual, aun siendo bastante críptico, con la práctica no resulta muy difícil de comprender. (Puede ver la lista de las instrucciones del lenguaje ensamblador en la Sección 28 de este capítulo.) Sin embargo, para ejecutar en el **+2A** un programa escrito en lenguaje ensamblador es necesario convertir esas instrucciones en una serie de bytes, que constituyen el programa en código de máquina. Esa conversión puede realizarla el propio ordenador mediante un programa que se llama *ensamblador*. En el **+2A** no hay ningún ensamblador incorporado, pero usted puede adquirirlo en cinta o disco. A falta de ensamblador, puede hacer la conversión usted mismo, aunque, en la práctica, esto sólo es posible con programas muy cortos.

Tomemos como ejemplo el programa:

ld bc, 99
ret

que carga el par de registros BC con el número 99. Al ensamblarlo, este programa se traduce en los cuatro bytes siguientes: 1, 99, 0 (que significan **ld bc, 99**) y 201 (que es **ret**). (Si consulta los códigos 1 y 201 en la tabla de la Sección 28, verá que el 1 corresponde a **ld bc,*NN***, donde *NN* representa cualquier número de dos bytes, y que el 201 corresponde a **ret**.)

Una vez convertido el programa de lenguaje ensamblador a código de máquina, el siguiente paso es introducirlo en la memoria del ordenador (un ensamblador probablemente lo haría de forma automática). Hay que empezar por decidir el lugar de la memoria en el que queremos situarlo (lo más conveniente es reservarle un espacio entre el área de BASIC y la de los gráficos definidos por el usuario).

La orden

CLEAR 65267

reserva un espacio de 100 bytes a partir de la dirección 65268.

Para introducir el programa de código de máquina, podríamos usar un programa de BASIC de este estilo:

```
10 LET a=65268
20 READ n: POKE a,n
30 LET a=a+1: GO TO 20
40 DATA 1,99,0,201
```

[Este programa se detiene y emite el mensaje '**E DATOS AGOTADOS**' cuando termina de escribir en la memoria los cuatro bytes especificados.]

Para ejecutar el programa de código de máquina se utiliza la función **USR** (pero ahora con argumento numérico; en concreto, la dirección inicial del programa). El resultado de **USR** es el valor que queda en el par de registros BC al retornar del programa en código de máquina, de manera que si escribimos

PRINT USR 65268

obtendremos la respuesta 99.

La dirección de retorno a BASIC es 'apilada' de la manera usual, así que el retorno se lleva a cabo mediante una instrucción **ret** del Z80. No debemos usar los registros IY e I en rutinas de código de máquina que vayan a utilizar el mecanismo de interrupciones de BASIC. Si el programa tiene que servir también para modelos anteriores de Spectrum (incluido el **+2**), tampoco debemos cargar I con valores comprendidos entre 40h y 7Fh (aunque no vayamos a usar nunca IM 2). Los valores entre C0h y FFh para I también deberían ser eludidos si se va a insertar la memoria compartida (RAM 4 o RAM 7) entre C000h y FFFFh. Esto se debe a una interacción entre el controlador de video y el mecanismo de refresco del Z80, que puede ocasionar la caída del sistema, la corrupción de la pantalla u otros efectos indeseables. Por tanto, sólo debemos dirigir las interrupciones IM 2 hacia direcciones comprendidas entre 8000h y BFFFh, a menos que estemos muy seguros de la estructura del mapa de memoria o solamente vayamos a usar el programa en el **+2A**, donde este problema no existe.

En la documentación de modelos anteriores (salvo en la del **+3**) se decía que la variable de sistema que está en 5CB0h (23728) no se utilizaba. En el **+2A** y en el **+3** esta variable es un vector de salto para interrupciones no enmascarables (NMI). Si se produce una NMI, el sistema comprueba el valor de estos bytes; si es 0, no hace nada; si es distinto de cero, salta a la posición especificada por esta variable. No debe producirse ninguna NMI mientras el sistema de disco esté activo.

Hay una serie de problemas típicos que se presentan sistemáticamente cuando se programa en código de máquina un sistema de conmutación de bancos como el del **+2A**. Si a usted le ocurre lo mismo, compruebe que su pila no está siendo conmutada durante las interrupciones, y que su rutina de interrupción está siempre donde debe estar (es conveniente inhibir las interrupciones durante las operaciones de paginación). También es recomendable que guarde una copia del registro de bancos actual en algún lugar no paginado de la RAM, ya que la puerta es de sólo escritura. BASIC y el editor usan las variables de sistema BANKM y BANK678 para 7FFDh y 1FFDh, respectivamente.

Si hace llamadas a +3DOS, no olvide que las interrupciones tienen que estar habilitadas al entrar en las rutinas. Recuerde también que la pila tiene que estar por debajo de BFE0h (49120) y por encima de 4000h (16384), y que tiene que haber al menos 50 palabras de espacio para la pila.

El programa de código de máquina se puede grabar con toda facilidad:

SAVE "*nombre*" CODE 65268,4

No hay ninguna manera de grabar el programa de forma que se ejecute automáticamente a sí mismo una vez cargado. No obstante, esto se remedia empleando un pequeño programa de BASIC:

```
10 LOAD "nombre" CODE 65268,4
20 PRINT USR 65268
```

que debe ser grabado como programa independiente (si es en la cinta, inmediatamente antes que el código de máquina) con la orden:

SAVE "cargador" LINE 10

Hecho esto, se puede ejecutar el código de máquina desde BASIC con sólo dar la orden:

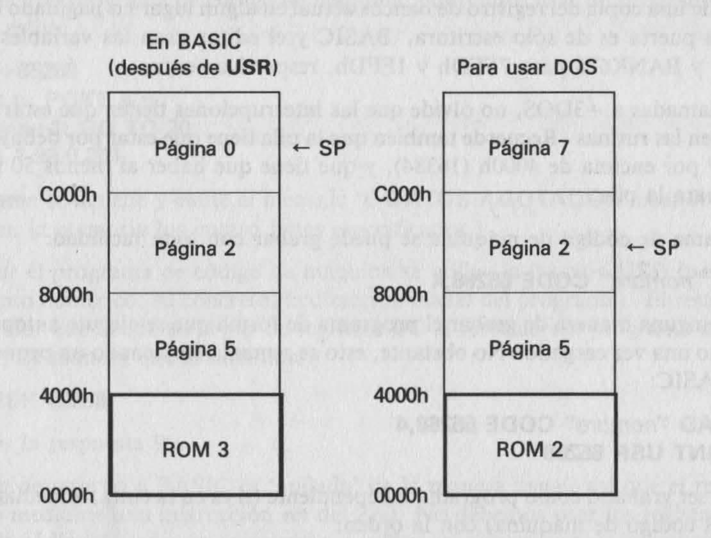
LOAD "cargador"

que carga y ejecuta automáticamente el programa de BASIC, el cual a su vez carga y ejecuta el código de máquina.

Llamadas a +3DOS desde BASIC

Al utilizar la función **USR** de BASIC, éste transfiere el control a la rutina de código de máquina especificada; la memoria está entonces configurada según se indica en la siguiente figura (a la izquierda). La ROM encajada en la zona inferior de la memoria (0000h a 3FFFh) es ROM 3 (la de 48 BASIC). La página de RAM encajada en el extremo superior es la página 0; la pila de máquina reside en esta zona (a menos que se la haya trasladado con una orden **CLEAR** a alguna posición por debajo de C000h). Según veremos en la Sec-

ción 27 (en la que están descritas las rutinas de +3DOS), para poder invocar +3DOS es necesario que la página 7 de RAM se encuentre en la zona superior de la memoria, que la pila se encuentre entre 4000h y BFE0h y que ROM 2 (que es la ROM de DOS) esté en el extremo inferior de la memoria. Esta configuración es la ilustrada en la siguiente figura (a la derecha).



Por consiguiente, habrá que conmutar tanto la ROM como la RAM y trasladar la pila antes y después de hacer cualquier llamada a la tabla de saltos de DOS. En el siguiente ejemplo ilustramos la forma de hacerlo (para invocar la rutina DOS CATALOGO).

Si se ha ejecutado desde BASIC una orden **CLEAR** para poner la pila por debajo de BFE0h (49120), ya no será necesario volver a trasladarla. En nuestro ejemplo sí la trasladaremos, para ilustrar cómo es el procedimiento.

Ejemplo de llamada a DOS CATALOGO

```
            org  7000h

mipila          equ 9FFFh    ;valor arbitrario, inferior a BFE0h y superior a 4000h
almpila         equ 9000h    ;un sitio para el puntero de la pila de BASIC
bankm           equ 5B5Ch    ;variable de sistema que contiene el último valor enviado a 7FFDh
puerta1         equ 7FFDh    ;dirección de la puerta de conmutación de ROM y RAM en el mapa de E/S
tampcat         equ 8000h    ;un sitio para que DOS deposite el catálogo
dos_catalogo    equ 011Eh    ;dirección de la rutina de DOS que vamos a invocar

demo:

        di                      ;no debemos conmutar ROM ni RAM sin inhibir interrupciones
        ld      (almpila),sp    ;guardar puntero de pila de BASIC
        ld      bc,puerta1      ;dirección de puerta de conmutación de RAM y de ROM (horizontal)
        ld      a,(bankm)       ;variable de sistema que contiene el estado de conmutación actual
        res     4,a             ;desplazar de derecha a izquierda en el conmutador horizontal de ROM (de 3 a 2)
        or      7               ;encajar página 7 de RAM
        ld      (bankm),a       ;es muy importante mantener actualizada esta variable de sistema
        out     (c),a           ;realizar la conmutación
        ld      sp,mipila       ;tiene que estar por encima de 4000h y por debajo de BFE0h

;
; La rutina anterior ha encajado la ROM de DOS y la página 7 de RAM, y además ha puesto la pila en un lugar
; seguro para poder invocar el sistema operativo
;
; Lo que sigue prepara las cosas para invocar DOS_CATALOGO y hace la llamada. Aquí es donde usted pondría su
; programa.
;
        ld      hl,tampcat      ;el sitio donde DOS nos entregará el catálogo
        ld      de,tampcat+1
        ld      bc,1024         ;el máximo es en realidad 64×13+13 = 845
        ld      (hl),0
        ldir                    ;para que al menos la primera reseña esté a cero
        ld      b,64            ;número de reseñas del tampón
        ld      c,1             ;incluir ficheros de sistema en el catálogo
        ld      de,tampcat      ;dirección a partir de la que rellenaremos con el catálogo del disco
        ld      hl,asteaste     ;es el nombre de fichero ("*.*")
        call    dos_catalogo    ;hacer la llamada a la rutina de DOS
        push    af              ;guardar indicadores y el posible código de error generado por DOS
        pop     hl
        ld      (dosret),hl     ;ponerlo donde BASIC pueda verlo
        ld      c,b             ;trasladar el número de ficheros al byte bajo de BC
        ld      b,0             ;esto es lo que nos entregará la función USR en BASIC

;
; Si todo ha ido bien, BC contiene el número de ficheros que hay en el catálogo; la zona que empieza en tampcat
; contiene el catálogo alfanumérico, ordenado alfabéticamente; el indicador de arrastre de dosret estará a 1.
; Esto lo leeremos en BASIC para ver si la rutina ha terminado bien.
;
; Después de hacer la llamada a DOS tenemos que dejar la RAM y la ROM como estaban y poner BASIC en el estado en
; que se encontraba al salir de él. Para ello necesitamos las siguientes instrucciones:
;
        di                      ;porque vamos a conmutar la RAM y la ROM
        push    bc              ;guardar número de ficheros
        ld      bc,puerta1      ;dirección del conmutador horizontal
        ld      a,(bankm)       ;leer el estado de conmutación actual
        set     4,a             ;desplazar de izquierda a derecha en el conmutador horizontal de ROM (de 2 a 3)
        and     F8h             ;encajar página 0 de RAM
        ld      (bankm),a       ;actualizar la variable de sistema (muy importante)
        out     (c),a           ;realizar la conmutación
        pop     bc              ;recuperar el número de ficheros
        ld      sp,(almpila)    ;restablecer la pila de BASIC
        ret                     ;retornar a BASIC (el valor que tenemos en BC es el que entregará USR)
```

```
asteaste:
defb        "*.*",FFh    ;nombre de fichero, que tiene que terminar en FFh
dosret:
defw        0            ;variable que leeremos en BASIC para ver si la rutina ha tenido éxito
```

En beneficio de los lectores que no dispongan de un programa ensamblador, facilitamos el siguiente programa de BASIC, que coloca los bytes del programa anterior en la memoria, lo ejecuta y finalmente utiliza el valor entregado por **USR** y el contenido de 'dosret' para escribir un sencillo catálogo del disco:

```
 10 LET suma=0
 20 FOR i=28672 TO 28758
 30 READ n
 40 POKE i,n : LET suma=suma+n
 50 NEXT i
 60 IF suma <> 9387 THEN PRINT "Hay un error en las líneas DATA" : STOP
 70 LET x=USR 28672
 80 IF INT ( PEEK (28757)/2)= PEEK (28757)/2 THEN PRINT "Error de
    disco "; PEEK (28758) : STOP
 90 IF x=1 THEN PRINT "No hay ficheros": STOP
100 FOR i=0 TO x-2
110 FOR j=0 TO 10
120 PRINT CHR$ ( PEEK (32781+i*13+j));
130 NEXT j
140 PRINT
150 NEXT i
160 DATA 243,237,115,0,144,1,253,127,58,92,91,203,167,246,7,
    50,92,91,237,121,49,255,159,251
170 DATA 33,0,128,17,1,128,1,0,4,54,0,237,176,6,64,14,1,17,0,
    128,33,81,112,205,30,1,245,225,34,85,112,72,6,0
180 DATA 243,197,1,253,127,58,92,91,203,231,230,248,50,92,91,
    237,121,193,237,123,0,144,201
190 DATA 42,46,42,255,0,0
```

Las direcciones que hemos elegido para el programa anterior y sus datos son enteramente arbitrarias. No obstante, es buena idea limitarse a los 32K centrales siempre que sea posible, con objeto de eludir el riesgo de que alguna variable o algún trozo del programa se pierdan al ser «desconmutados».

Si las interrupciones tienen que estar habilitadas (como ocurría en el ejemplo anterior), es fundamental mantener el sistema informado acerca de la conmutación de ROM más reciente. En concreto, compete al usuario conseguir que la variable de sistema BANK678 refleje siempre el último byte enviado a la puerta 1FFDh. Tal como ilustra el ejemplo an-

terior, la técnica consiste en copiar la variable en el acumulador, modificar los bits relevantes, actualizar la variable de sistema y realizar la conmutación con una instrucción **OUT**. Las interrupciones tienen que estar inhibidas en todo momento en que la variable de sistema no refleje el estado de la puerta. Puesto que la puerta 1FFDh no se limita a controlar la conmutación de las ROM, sería incorrecto dar a la variable valores absolutos; lo que se debe hacer es modificar solamente los bits pertinentes, utilizando instrucciones **SET/RES** o bien **AND/OR** con la máscara de bits adecuada.

De la misma forma que BANK678 refleja el valor enviado a 1FFDh, BANKM debe estar permanentemente actualizada con el último valor enviado a la puerta 7FFDh. Tampoco en este caso se debe usar valores absolutos, pues ciertos bits de esta puerta sirven para otros propósitos. Por ejemplo, los 3 bits inferiores seleccionan la página de RAM que es encajada en la zona de C000h a FFFFh (como se puede observar en el ejemplo anterior). Desde luego, cuando haya que controlar varios bits a un tiempo, la forma más rápida es usar **AND** u **OR** con la máscara adecuada. El sistema de paginación de la RAM está descrito en la Sección 24, en el apartado 'Gestión de la memoria'.

El ejemplo anterior demostraba la forma más sencilla de invocar rutinas de DOS. En el siguiente mostraremos un par de técnicas adicionales que pueden ser muy útiles. Sin embargo, si usted no tiene cierta soltura en la programación en ensamblador, quizá sea preferible que ignore este ejemplo.

El programa anterior funciona aunque no se tenga conectada la unidad de disquete externa; produce el catálogo de la unidad M (que será la unidad implícita). En cambio, el programa siguiente sólo funcionará si se ha conectado la unidad externa (y el interfaz).

En la Sección 20 de este capítulo dijimos que la opción **Cargador** del menú de presentación primero busca en el disco el fichero llamado *, después el fichero **DISK** y finalmente, si no encuentra ninguno de los dos, carga el primero que encuentra en la cinta. Pero ésta no es la historia completa. En realidad, la primera operación consiste en intentar cargar un sector de *inicialización* desde el disco de la unidad A. El sistema utiliza como cargador de inicialización el sector de la cara 0 y pista 0, a condición de que la suma de comprobación de este sector sea 3. El siguiente programa asegura que la suma de comprobación de los 512 bytes cumple este requisito y luego escribe la información en la posición correcta en el disco. Una vez modificado el disco de esta forma, se puede usar la opción **Cargador** para cargarlo y ejecutarlo automáticamente. También se puede usar la orden **LOAD "*"** de BASIC.

Este ejemplo ha sido desarrollado con la ayuda de M80 en una máquina basada en CP/M; el método por el que se asegura que el programa ensamblado quede referido a la dirección correcta puede ser distinto en otros ensambladores.

```
;
; Este programa escribe un sector de inicialización en el disco de la unidad A
;
; escrito por Cliff Lawson
; copyright (c) AMSTRAD Plc. 1987
;

        .z80                    ;ignore esto si su ensamblador no es M80
```

```
banco1          equ 07FFDh ;puerta de paginación de RAM y de conmutación horizontal de ROM
bankm           equ 05B5Ch ;variable de sistema asociada
banco2          equ 01FFDh ;puerta de conmutación vertical de ROM
bank678         equ 05B67h ;variable de sistema asociada

selecc          equ 01601h ;rutina de BASIC para abrir canales
dos_ref_xdpb    equ 0151h  ;
dd_escr_sector  equ 0166h  ;véase la Sección 27
dd_reconocer    equ 0175h  ;

        org     0
        .phase  07000h

;
; (Esto hace que M80 genere un fichero .COM cuyas direcciones están referidas a 7000h. Se ensambla
; con "M80 = prog" y se enlaza con "L80 /p:0,/d:0,prog,prog/n:p/y/e". A este fichero se le puede poner una
; cabecera con COPY...TO SPECTRUM FORMAT y entonces se lo puede cargar con LOAD...CODE 28672.
;
; Posiblemente se requerirá otra técnica si se trabaja con otros ensambladores.)
;
empezar:
        ld      (pilaantig),sp      ;guardar el puntero de pila de BASIC
        ld      sp,mipila           ;poner la pila por debajo de las páginas de RAM conmutables
        push    iy                  ;guardar IY en la pila por el momento
        ld      a,"A"               ;unidad A
        ld      iy,dos_ref_xdpb     ;IX tiene que apuntar al XDPB de A (necesario para invocar rutinas de DOS)
        call    irdos
        ld      c,0                 ;reconocer el disco de la unidad 0 para que al escribir sectores DOS no nos diga
        push    ix                  ;que hemos cambiado el disco
        ld      iy,dd_reconocer
        call    irdos
        pop     ix

        ld      hl,sectorinic
        ld      bc,512              ;vamos a hacer la suma de comprobación de los 512 bytes del sector
        xor     a
        ld      (sectorinic+15),a   ;inicializamos la suma
        ld      e,a                 ;E contendrá la suma en 8 bits

buclesuma:
        ld      a,e
        add     a,(hl)              ;este bucle hace la suma de comprobación (en 8 bits) de los 512 bytes del sector
        ld      e,a                 ;y la va depositando en E
        inc     hl
        dec     bc
        ld      a,b
        or      c
        jr      nz,buclesuma

        ld      a,e                 ;A contiene la suma en 8 bits
        cpl                         ;complemento a uno (+1 dará valor negativo)
        add     a,4                 ;sumar 3 para hacer suma igual a 3, +1 para hacer complemento a dos
        ld      (sectorinic+15),a   ;hace que la suma de los bytes sea 3 al tomarla módulo 256

        ld      b,0                 ;página 0 en C000h
        ld      c,0                 ;unidad 0
        ld      d,0                 ;pista 0
        ld      e,0                 ;sector físico 1 (o sea, sector lógico 0)
        ld      hl,sectorinic       ;dirección de la información que vamos a escribir en el sector de inicialización
        ld      iy,dd_escr_sector
        call    irdos               ;escribir el sector en el disco
        pop     iy                  ;restablecer IY para que BASIC pueda hacer referencia a sus variables de sistema
        ld      sp,(pilaantig)      ;restablecer la pila original
        ret                         ;retornar a la USR de BASIC
```

```
irdos:
;
; IY contiene la dirección de la rutina de DOS que va a ser ejecutada. Todos los demás registros son entregados
; intactos a la rutina de DOS, y ésta los devuelve modificados.
;
; La pila está en algún lugar de los 32K centrales (de acuerdo con los requisitos de DOS), de modo que al guardar
; AF y BC no hay riesgo de perderlos en una operación de conmutación.
;
        push    af
        push    bc              ;guardar estos registros temporalmente durante la conmutación
        ld      a,(bankm)       ;variable que controla la conmutación
        or      7               ;porque queremos la página 7 de la RAM
        res     4,a             ;y la ROM de DOS
        ld      bc,banco1       ;puerta de paginación de RAM y de conmutación horizontal de ROM
        di
        ld      (bankm),a       ;para mantener actualizada la variable de sistema
        out     (c),a           ;página 7 de la RAM arriba y ROM de DOS
        ei
        pop     bc
        pop     af

        call    puntosalto      ;ir a la subrutina cuya dirección está en IY

        push    af              ;el retorno después de JP (IY) nos traerá hasta aquí
        push    bc
        ld      a,(bankm)
        and     0F8h            ;poner a 0 los bits para página 0
        set     4,a             ;encajar ROM 3 (la de 48 BASIC)
        ld      bc,banco1
        di
        ld      (bankm),a
        out     (c),a           ;otra vez página 0 de la RAM y ROM de 48 BASIC
        ei
        pop     bc
        pop     af
        ret

puntosalto:
        jp      (iy)            ;forma estándar de ejecutar CALL (IY) haciendo una llamada este punto

pilaantig:
        dw      0               ;un sitio para el puntero de la pila de BASIC
        ds      100

mipila:                         ;suficiente para cumplir los requisitos de +3DOS

sectorinic:
    .dephase                    ;son pseudooperadores de M80
    .phase      0FE00h          ;otros ensambladores pueden utilizar algo diferente

;
; El cargador de inicialización será cargado en la página 3, a partir de FE00h. El punto de entrada (dirección de
; ejecución) será FE10h.
;
; Antes de escribirlo en el disco (en la pista 0, sector 1), hemos ajustado el byte 15 del cargador para que su
; suma de comprobación sea 3 al tomarla módulo 256.
;
; El cargador conmuta la memoria de modo que la ROM de 48 BASIC quede en el extremo inferior. Por encima estará
; la página 5 (pantalla), después la página 2 y, arriba del todo, permanece la página 3, pues no sería correcto
; desconmutar el cargador. Las rutinas de BASIC pueden ser invocadas cualquiera que sea la página que tengamos
; arriba, pero la pila no estaría en la zona de TSTACK.
;

princcarg:
;
```

```
; El sector de carga de inicialización lleva al principio, en 16 bytes, la especificación del disco. Los valores
; siguientes corresponden al formato AMSTRAD PCW CF2/Spectrum +3.
;
        db      0               ;formato del +3
        db      0               ;una sola cara
        db      40              ;40 pistas por cara
        db      9               ;9 sectores por pista
        db      2               ;log2(512)-7, tamaño de sector
        db      1               ;1 pista reservada
        db      3               ;bloques
        db      2               ;2 bloques de directorio
        db      02Ah            ;longitud de intervalo (lectura/escritura)
        db      052h            ;longitud de intervalo (formato)
        ds      5,0             ;5 bytes reservados

suma:   db      0               ;la suma de comprobación (módulo 256) tiene que ser 3 para el sector
;
; Al entrar en el cargador estarán encajadas las páginas 4, 7, 6 y 3 de la RAM.
; Para poder escribir algo necesitamos 48 BASIC abajo y la página 5 (pantalla y variables de sistema)
; inmediatamente por encima. La siguiente página será la 0, y la de arriba seguirá siendo la 3 porque contiene el
; cargador y la pila (ésta se encuentra en FE00h a la entrada).
;
        di
        ld      a,(bankm)
        and     0F8h
        or      3               ;página 3 de la RAM (porque contiene el cargador)
        set     4,a             ;ROMs de la derecha
        ld      bc,banco1
        ld      (bankm),a
        out     (c),a           ;conmutar ROM en horizontal y RAM
        ld      a,(bank678)
        and     0F8h
        or      4               ; bit 2 a 1 y bit 0 a 0 (para ROM 3)
        ld      bc,banco2
        ld      (bank678),a
        out     (c),a
        ld      a,2
        call    selecc          ;rutina de la ROM de BASIC para abrir un canal
        ld      hl,mensaje
        call    escribir        ;escribir un mensaje

buclefin:
;
; Terminamos con un bucle sin fin que cambia el color del borde. Aquí es donde pondríamos un programa de juego,
; un sistema operativo, etc.
;
        ld      a,r             ;un número no muy aleatorio
        out     (0feh),a        ;cambiar el borde
        jr      buclefin        ;y otra vez al principio del bucle

escribir:
        ld      a,(hl)          ;esto es un bucle que imprime caracteres
        cp      0FFh            ;hasta que encuentra FFh
        ret     z
        rst     10h             ;con la ROM de 48 BASIC, esto escribe el carácter que está en el acumulador
        inc     hl
        jr      escribir

mensaje:
        defb    16,2,17,7,19,0,22,10,1,"Hola, buenas noches, bienvenido",0ffh

cliff:
        ds      512-(cliff-princcarg),0   ;rellenar con ceros hasta el final del sector
        end
```

Hay algunos detalles en este ejemplo que vale la pena resaltar. En primer lugar, dado que BASIC normalmente tiene la dirección de la variable de sistema ERR_NR en IY (para facilitar las referencias a sus variables de sistema), es imprescindible guardar IY antes de hacer cualquier otra cosa y luego restaurarlo antes de retornar a la **USR** original.

Al igual que antes, hemos trasladado la pila a los 32K centrales. De esta manera podemos invocar las rutinas de +3DOS sin tener que volver a trasladarla.

La rutina 'irdos' puede ser útil en cualquier programa. Sólo utiliza el registro IY, que el sistema +3DOS no necesita, y puede hacer llamadas a cualquier rutina de +3DOS.

El programa invoca DOS REF XDPB para hacer que IX apunte al XDPB de la unidad A. Después 'reconoce' el disco de A, paso imprescindible para poder escribir en él. Luego calcula y ajusta la suma de comprobación de los bytes que van a ser transferidos al sector de carga del disco. Y finalmente invoca DD ESCR SECTOR para escribir el sector.

Por no complicar las cosas, ni siquiera hemos comprobado que existe el interfaz de disco, y hemos ignorado los posibles errores. Esta rutina puede ser puesta en marcha desde BASIC con la orden:

USR 28672

la cual entrega, a su retorno, el número que haya quedado en BC.

Los primeros 16 bytes del sector de carga constituyen una especificación de disco estándar. A continuación viene el código de máquina que forma el cargador de inicialización, cuyo punto de entrada es FE10h. Tal como explicaremos en la Sección 27 a propósito del interfaz para DOS CARGAR, la memoria consistirá inicialmente en las páginas 4, 7, 6, 3; las variables de sistema de BASIC permanecen intactas, de modo que se puede usar BASIC encajando la ROM correcta (la 3) en la parte inferior de la memoria y asegurándose de que la página 5 está entre 4000h y 7FFFh.

Este sencillo programa cargador no hace más que usar la ROM de BASIC para escribir un mensaje y luego entrar en un bucle que cambia el color del borde de la pantalla. Usted podría modificarlo, por ejemplo, para que cargase un fichero binario e iniciase su ejecución.

Sección 27
Guía de +3DOS

Temas tratados:

ROMs
Interfaz de +3DOS
Atributos y cabeceras de ficheros
Formato y especificación de los discos
Pistas y sectores
Bloques de parámetros de disco
Compatibilidad de ficheros de CP/M
Cambio de disco
Proyección de unidades lógicas a unidades físicas
Mensajes y requisitos de +3DOS
Rutinas de +3DOS

Gran parte de la información que daremos en esta sección solamente es válida para los equipos en los que se haya conectado una o dos unidades de disquete externas.

En esta sección vamos a describir +3DOS, que es el sistema operativo de disco del **+2A**. Es probable que esta información sólo interese a los lectores que sepan programar en ensamblador (código de máquina); véase la Sección 26, en la que está tratado ampliamente este tema.

Aunque no exista la unidad externa, muchas de las rutinas de +3DOS son aplicables a la unidad M (disco de RAM). Si se ha conectado la unidad externa (con el interfaz adecuado), **todas** las rutinas de +3DOS estarán disponibles. Véase el apartado 'Utilización de +3DOS en ausencia del interfaz de disco'.

El 'software' del **+2A** se encuentra en cuatro ROMs (si bien toda la información está en sólo dos circuitos integrados). Las cuatro ROMs han de ser «direccionadas» en el margen de 0000h a 3FFFh, pero sólo una estará disponible en un momento dado.

La ROM 0 es la 'ROM del editor'. Es la seleccionada cuando se acaba de encender el ordenador. Esta ROM controla los menús y las funciones de edición de alto nivel.

La ROM 1 es la 'ROM de sintaxis'. Es la encargada de realizar el control de +3BASIC a alto nivel. Contiene el código que realiza la parte correspondiente a BASIC en la mayor parte de las órdenes de manejo del disco.

La ROM 3 es la 'ROM de 48 BASIC', prácticamente idéntica a la del primer Spectrum. La única diferencia radica en la rutina que se ejecuta en respuesta a una interrupción. Hay una variable que, mientras sea distinta de cero, es decrementada cada dos interrupciones; cuando llega a cero, se desconecta el motor del disco. Esta variable se encuentra en la página 7, junto con algunas variables del editor y de DOS. La página 7 sólo es encajada (y la variable es decrementada) cuando el bit 4 de la variable de sistema FLAGS está a 1; de esta manera se identifica si el ordenador está ejecutando 48 BASIC o +3BASIC. Si está activo 48 BASIC (desde el menú de presentación o como consecuencia de una orden **SPECTRUM**) este bit está a 0, y entonces no se produce el decremento de la variable ni la conmutación de páginas. Sin embargo, si más tarde el programa de usuario pone a 1 el bit 4 de la variable FLAGS, ese proceso se pondrá en marcha, incluso con modo de interrupción 1 aún seleccionado.

Las rutinas de exploración del teclado, que se encontraban en la ROM 3 en el Spectrum 128 y en el **+2**, han sido trasladadas al **+2A**.

Un error que había en la ROM de 48 BASIC original ha sido corregido en el **+2A**. Cuando ocurre una interrupción no enmascarable (NMI), se ejecuta un salto a la dirección 66h. En ésta se comprueba el contenido de la variable de sistema NMIADD. Si es cero, se ejecuta un RETN; de lo contrario, se salta a la dirección indicada por NMIADD. En la ROM 2 los códigos para NMI consisten en un simple RETN.

La ROM 3 no sólo proporciona el modo 48 BASIC por razones de compatibilidad con modelos anteriores, sino que ejecuta la mayor parte de las órdenes de +3BASIC que no utilizan el hardware más avanzado del **+2A**.

La última ROM (ROM 2) contiene +3DOS, que es el sistema operativo de disco. Esta ROM es el objeto de la presente sección. A diferencia de las otras ROMs, que no tendrán mucha utilidad para los programadores de lenguaje ensamblador (salvo quizá la de 48 BASIC), la ROM de +3DOS proporciona gran número de rutinas que pueden ser aprovechadas en los programas de usuario. Es muy deseable que los programas que utilicen las unidades de disco se sirvan de estas rutinas, con las que se puede realizar todas las funciones imaginables (de hecho, más que las que aprovecha la versión actual de BASIC). Por otra parte, no se debe acceder a las rutinas más que a través del bloque de saltos. Al hacerlo así, los programas serán más fácilmente intercambiables con los de los ordenadores AMSTRAD CPC y, además, compatibles con los Spectrum futuros. Los puntos de entrada a las diversas rutinas están en una tabla de saltos que empieza en la dirección 0100h (256) de la ROM. En la Sección 26 de este capítulo hemos dado un par de ejemplos que ilustran la forma de invocar las rutinas de DOS.

+3DOS proporciona rutinas para realizar las siguientes funciones:

- Gestión de una o dos unidades de disquete y de un disco de RAM.
- Compatibilidad con ficheros de CP/M Plus y de CP/M 2.2.
- Compatibilidad con ficheros y discos de los ordenadores AMSTRAD, series CPC y PCW.
- Posibilidad de tener abiertos hasta 16 ficheros a un tiempo.

- Lectura y escritura de ficheros hacia (o desde) cualquier página de la memoria.
- Acceso directo (aleatorio) a nivel de byte.
- Borrado de ficheros de disco; cambio de nombre de los ficheros; cambio de los atributos de los ficheros.
- Selección de la unidad y del número de usuario implícitos (por defecto).
- Carga de un juego o de un sistema operativo.
- Acceso a bajo nivel al controlador de la unidad de disquete.
- Proyección opcional de dos unidades lógicas (A y B) hacia una sola unidad física (unidad 0).

Interfaz de +3DOS

El interfaz de +3DOS es un conjunto de rutinas a las que se accede a través de un bloque de saltos. Estas rutinas están divididas en tres grupos:

- Rutinas esenciales del sistema de archivo.
- Rutinas adicionales para juegos y sistemas operativos.
- Rutinas de control del disco a bajo nivel (para formatear, copiar, etc.).

Las siguientes tablas dan la lista de las rutinas de los tres grupos, junto con una breve descripción de sus respectivas funciones.

Rutinas esenciales del sistema de archivo

Nombre de la rutina	Función
DOS INICIALIZAR	Inicializar el sistema operativo +3DOS
DOS VERSION	Averiguar los números de edición y de versión de +3DOS
DOS ABRIR	Crear y/o abrir un fichero
DOS CERRAR	Cerrar un fichero
DOS ABANDONAR	Abandonar un fichero
DOS REF CAB	Apuntar a los datos de cabecera de un fichero
DOS LEER	Leer bytes y copiarlos en la memoria
DOS ESCRIBIR	Copiar bytes desde la memoria hacia el disco
DOS LEER BYTE	Leer un byte
DOS ESCR BYTE	Escribir un byte
DOS CATALOGO	Catálogo del directorio del disco
DOS ESPACIO	Espacio que queda libre en el disco
DOS BORRAR	Borrar un fichero
DOS CAMB NOMBRE	Cambiar el nombre de un fichero
DOS CARGAR	Cargar un sistema operativo u otro programa
DOS EST UNIDAD	Establecer la unidad implícita o averiguar cuál es
DOS EST USUARIO	Establecer el número de usuario implícito o averiguar cuál es

Rutinas adicionales para juegos y sistemas operativos

Nombre de la rutina	Función
DOS LEER POS	Averiguar posición del puntero del fichero para acceso directo (aleatorio)
DOS EST POS	Establecer posición del puntero del fichero para acceso directo (aleatorio)
DOS LEER EOF	Averiguar posición del final del fichero para acceso directo (aleatorio)
DOS LEER 1346	Averiguar la utilización de la memoria en las páginas 1, 3, 4, 6
DOS EST 1346	Reasignar la utilización de la memoria en las páginas 1, 3, 4, 6
DOS ACTUALIZAR	Actualizar el disco
DOS EST ACCESO	Cambiar el modo de acceso de un fichero abierto
DOS EST ATRIB	Cambiar los atributos de un fichero
DOS ABRIR UNIDAD	Abrir una unidad como un único fichero
DOS EST MENSAJE	Habilitar o inhibir los mensajes de error
DOS REF XDPB	Apuntar al XDPB para el acceso a bajo nivel
DOS PROYEC B	Proyectar la unidad B hacia la 0 o la 1

Rutinas de control del disco a bajo nivel

Nombre de la rutina	Función
DD INTERFAZ	Averiguar si está presente el interfaz controlador de la unidad de disquete
DD INIC	Inicializar el controlador
DD CONFIGURAR	Especificar los parámetros de la unidad
DD EST REINT	Inicializar el contador de 'reintentar'
DD LEER SECTOR	Leer un sector
DD ESCR SECTOR	Escribir un sector
DD VERIF SECTOR	Verificar un sector
DD FORMATEAR	Formatear una pista
DD LEER ID	Leer el identificador de un sector
DD PROBAR DISCO	Averiguar si el disco es de tipo adecuado
DD RECONOCER	Reconocer el disco, inicializar el XDPB
DD SELEC FORMATO	Preinicializar XDPB para DD FORMATEAR

Nombre de la rutina	Función
DD BUSCAR 1	Averiguar si está presente la unidad 1 (externa)
DD ESTADO UNIDAD	Averiguar el estado de la unidad
DD EQUIPO	Averiguar de qué tipo es la unidad
DD PROT	Establecer la rutina de intercepción para la protección contra copias
DD L XDPB	Inicializar un XDPB a partir de una especificación de disco
DD L DPB	Inicializar un DPB a partir de una especificación de disco
DD L BUSCAR	Buscar una pista
DD L LEER	Orden de lectura de bajo nivel del μPD765A
DD L ESCRIBIR	Orden de escritura de bajo nivel del μPD765A
DD L MOTOR MARCHA	Poner en marcha el motor, esperar hasta que transcurra el 'tiempo de puesta en marcha'
DD L MOTOR TEMP	Iniciar el descuento del 'tiempo de desconexión' del motor
DD L MOTOR DESC	Desconectar el motor

Juegos y otros programas de código de máquina

+3DOS proporciona diversas funciones específicas para los programas no escritos en BASIC:

- Usar DOS CARGAR para cargar un sector de inicialización, el cual toma entonces el control de toda la máquina.
- Preservar espacio para +3DOS mediante DOS EST 1346. De esta manera el programa de usuario puede controlar toda la máquina y seguir disponiendo de las funciones de +3DOS cuando las necesite. Si el programa no requiere los servicios de +3DOS, puede ejecutar DD L MOTOR TEMP para forzar la parada del motor e inhibir su temporizador. Se debe poner a cero el bit 4 de la variable de sistema FLAGS para impedir la conmutación de bancos y la modificación de variables en caso de interrupción.
- Es posible abrir una unidad para usarla como si fuera un solo fichero. De esta manera se puede examinar los ficheros y los directorios sin tener que pasar por la estructura de ficheros.

Utilización de +3DOS en ausencia del interfaz de disco

A pesar de que el **+2A** no incluye ningún interfaz controlador de disco, se puede usar +3DOS (con las siguientes limitaciones):

- Sólo está disponible la unidad M (disco de RAM).
- La unidad implícita para las especificaciones de fichero es inicialmente M (en lugar de A).
- Todo intento de usar las unidades A y B fracasa y provoca el mensaje de error **'22 UNIDAD NO ENCONTRADA'**.
- Puesto que el disco de RAM no requiere el uso de un caché de sector, el tamaño de ese disco aumenta hasta 64K (las páginas 1, 3, 4 y 6 completas), que se reparten en 62K para datos y 2K para el directorio (64 reseñas).
- La presencia del interfaz de disco se detecta mediante DD INTERFAZ. Si está ausente, no se debe ejecutar ninguna de las restantes rutinas de bajo nivel (DD ...), pues su resultado es imprevisible.

Ahora bien, si se ha conectado una unidad de disquete externa (con el interfaz adecuado), +3DOS no está sometido a ninguna de estas limitaciones.

Atributos de fichero

El bit 7 de los caracteres de los campos del nombre y del tipo contiene los atributos del fichero. Esos bits superiores de los 8 caracteres del nombre se denotan por f1...f8. Los bits superiores de los 3 caracteres del tipo son t1...t3. Su significado es como sigue:

f1...f4 — Disponibles para el usuario
f5...f8 — Reservados (siempre a 0)
t1 — 0 significa que el fichero es de lectura/escritura
1 significa que el fichero es de sólo lectura
t2 — 0 significa que el fichero no es de sistema
1 significa que el fichero es de sistema
t3 — 0 significa que el fichero no es de archivo
1 significa que el fichero es de archivo

Si un fichero es de sólo lectura, no se puede escribir en él, ni borrarlo, ni cambiarle el nombre. El atributo 'de archivo' es ignorado por +3DOS.

Los ficheros recién creados tienen todos los atributos a 0. Para modificar los atributos de un fichero se invoca DOS EST ATRIBUTOS (que es lo que hace la orden **MOVE** de BASIC).

Cabeceras de fichero

Los ficheros de cinta tienen una cabecera que contiene cierta información requerida por el sistema. Los ficheros de +3DOS pueden tener cabecera o no tenerla. Todos los ficheros creados por la orden **SAVE** de BASIC tienen cabecera.

El mecanismo de cabeceras de +3DOS reserva para BASIC un área de 8 bytes. El resto de la cabecera está reservado para +3DOS. El área de 8 bytes es utilizada por las órdenes de BASIC (véase la descripción de DOS ABRIR).

Los ficheros de +3DOS pueden tener una única cabecera en sus primeros 128 bytes (el *registro de cabecera*). Las cabeceras son detectadas por el sistema en virtud de una 'signatura' y una suma de comprobación. Si la signatura y la suma de comprobación son las esperadas, el sistema deduce que la cabecera está presente; de lo contrario, considera que no hay cabecera. Por consiguiente, es posible, aunque muy improbable, que un fichero sin cabecera sea tratado como si la tuviera.

El formato del registro de cabecera es como sigue:

Bytes 0...7	— Signatura de +3DOS (PLUS3DOS)
Byte 8	— 1Ah (26 decimal); EOF blando (fin de fichero por programa)
Byte 9	— Número de edición
Byte 10	— Número de versión
Bytes 11...14	— Longitud del fichero en bytes; número de 32 bits; el byte menos significativo está en la dirección más baja
Bytes 15...22	— Cabecera para +3BASIC
Bytes 23...126	— Reservados (todos a 0)
Byte 127	— Suma de comprobación (suma de los bytes 0 a 126 tomada módulo 256)

Los números de edición y de versión están previstos para ampliaciones futuras. El número de edición tendrá que coincidir con el del programa. El número de versión tendrá que ser igual o menor que el del programa.

+3DOS realiza todas las operaciones de mantenimiento de la cabecera. Ejecutando DOS REF CAB se obtiene un puntero que «apunta» hacia el área de 8 bytes de la cabecera de +3BASIC. Nunca es necesario escribir directamente en la cabecera.

Las cabeceras de AMSDOS (utilizadas por los ordenadores AMSTRAD de la serie CPC) no son reconocidas por +3DOS. Los ficheros de AMSDOS son tratados por +3DOS como si no tuvieran cabecera, y lo mismo hace AMSDOS con los ficheros de +3DOS.

Formatos de disco

+3DOS maneja exactamente los mismos formatos de disco que CP/M y LocoScript en los ordenadores AMSTRAD de la serie PCW.

+3DOS detecta los siguientes formatos en cuanto accede al disco por primera vez:

- AMSTRAD PCW, pista sencilla (v.g., el usado en el PCW8256)
- AMSTRAD PCW, pista doble (v.g., el usado en el PCW8512)
- AMSTRAD CPC, formato de sistema
- AMSTRAD CPC, formato comercial
- AMSTRAD CPC, formato de datos

Nótese que +3DOS no maneja del 'formato IBM' de los ordenadores AMSTRAD CPC.

El sistema puede gestionar otros formatos si se modifica el XDPB de una unidad. El XDPB es el correspondiente al primero de los formatos de la lista anterior; **no** es el mismo que en los AMSTRAD CPC.

En cualquier caso, los formatos están sujetos a las siguientes limitaciones:

- Tamaño de sector: 512 bytes
- Máximo de 255 sectores por pista
- Máximo de 255 pistas
- Máximo de 256 reseñas de directorio
- Máximo de 360 unidades de asignación

Pistas y sectores lógicos

Las rutinas del controlador de disco requieren pistas y sectores lógicos. De esta forma +3DOS puede ignorar todo lo relativo al número de caras y a los números de los sectores físicos.

En los discos de una sola cara, los números de las pistas lógicas coinciden con los números de las pistas físicas.

Para los discos de doble cara hay dos posibilidades:

1. Caras alternas:

   cara 0 pista 0 = pista lógica 0
   cara 1 pista 0 = pista lógica 1
   cara 0 pista 1 = pista lógica 2
   cara 1 pista 1 = pista lógica 3
   :
   cara 0 última pista = pista lógica $n-1$
   cara 1 última pista = pista lógica n

2. Caras sucesivas:

   cara 0 pista 0 = pista lógica 0
   cara 0 pista 1 = pista lógica 1
   cara 0 pista 2 = pista lógica 2
   :
   cara 0 última pista = pista lógica (*n*/2)−1
   :
   cara 1 última pista−1 = pista lógica (*n*/2)
   cara 1 última pista−2 = pista lógica (*n*/2)+1
   cara 1 última pista−3 = pista lógica (*n*/2)+2
   :
   cara 1 pista 0 = pista lógica *n*

donde *n* es el número total de pistas lógicas (o sea, 2 por el número de pistas por cara).

Los sectores lógicos ocultan los números de los sectores físicos. Los sectores lógicos siempre empiezan en 0:

   sector lógico = sector físico − primer sector físico

Especificación de los discos

El formato 'PCW' (usado en el **+3**) engloba, en realidad, una familia de formatos, de la que se elige uno concreto mediante la 'especificación de disco' grabada en los bytes 0...15, del sector 1, pista 0, cara 0. El formato utilizado por el **+3** (y por el **+2A**) es el del 'tipo de disco 0' de la tabla siguiente. El sector que contiene la especificación del disco puede ser usado también para almacenar un cargador de inicialización (véase a este respecto el segundo ejemplo de la Sección 26).

Byte 0	Tipo de disco: 0 = PCW estándar, DD UC PS (y **+3**) 1 = CPC estándar, DD UC PS, formato de sistema 2 = CPC estándar, DD UC PS, formato de datos 3 = PCW estándar, DD DC PD Todos los demás valores están reservados (DD=doble densidad, UC=una cara, DC=dos caras, PS=pista sencilla, PD=pista doble)
Byte 1	Bits 0...1, caras: 0 = una cara 1 = dos caras (alternas) 2 = dos caras (sucesivas) Bits 2...6, reservados (a 0) Bit 7, pista doble

Byte 2	Número de pistas por cara
Byte 3	Número de sectores por pista
Byte 4	Log_2(tamaño de sector)−7
Byte 5	Número de pistas reservadas
Byte 6	Log_2(tamaño de bloque/128)
Byte 7	Número de bloques de directorio
Byte 8	Longitud de intervalo (lectura/escritura)
Byte 9	Longitud de intervalo (formato)
Bytes 10...14	Reservados
Byte 15	Suma de comprobación (sólo utilizado cuando el disco es de autoarranque)

Cuando el sistema reconoce el disco, utiliza esta especificación para inicializar el XDPB pertinente.

Bloques de parámetros de disco ampliados (XDPB)

Asociado a cada unidad (lógica) hay un bloque de parámetros de disco ampliado (XDPB) que contiene, además del DPB estándar de los discos de CP/M Plus, la información que +3DOS necesita para manejar los diferentes formatos. Es posible modificarlo para usar discos de formato distinto (a condición de que se respete las limitaciones mencionadas en el apartado 'Formatos de disco').

Estructura de los XDPB:

Bytes 0...1	SPT, sectores por pista
Byte 2	BSH, log_2(tamaño de bloque/128)
Byte 3	BLM, (tamaño de bloque/128)−1
Byte 4	EXM, máscara de sección
Bytes 5...6	DSM, número del último bloque
Bytes 7...8	DRM, número de la última reseña del directorio

Byte 9	AL0, mapa de bits del directorio
Byte 10	AL1, mapa de bits del directorio
Bytes 11...12	CKS, tamaño del vector de suma de comprobación (bit 15=permanente)
Bytes 13...14	OFF, número de pistas reservadas
Byte 15	PSH, $\log_2$(tamaño de sector/128)
Byte 16	PHM, (tamaño de sector/128)−1
Byte 17	Bits 0...1, caras: 0 = una cara 1 = dos caras (alternas) 2 = dos caras (sucesivas) Bits 2...6, reservados (a 0) Bit 7, pista doble
Byte 18	Número de pistas por cara
Byte 19	Número de sectores por pista
Byte 20	Número del primer sector
Bytes 21...22	Tamaño de sector
Byte 23	Longitud de intervalo (lectura/escritura)
Byte 24	Longitud de intervalo (formato)
Byte 25	Bit 7, gestión de pistas: 1 = multi-pista 0 = pista sencilla Bit 6, modo de modulación: 1 = MFM 0 = FM Bit 5, saltar marca de dirección de datos borrada: 1 = saltar 0 = no saltar Bits 0...4 = 0
Byte 26	Indicador de congelación: 00h (0) = auto-detectar formato del disco FFh (255) = no auto-detectar formato del disco

El byte 25 es normalmente 60h (96). No se recomienda el funcionamiento en multi-pista.

El indicador de congelación (byte 26) se activa para impedir que +3DOS trate de averiguar el formato del disco. Esto interesa cuando se modifica el XDPB para formatos no estándar.

Los XDPB para los tres formatos principales son como sigue:

Formato AMSTRAD PCW de pista sencilla (tipo 0) (usado por el +3 y por la unidad externa del +2A)

36	SPT, sectores por pista
3	BSH, desplazamiento de bloque
7	BLM, máscara de bloque
0	EXM, máscara de sección
174	DSM, número de bloques − 1
63	DRM, número de reseñas del directorio − 1
C0h (192)	AL0, 2 bloques de directorio
00h (0)	AL1
16	CKS, tamaño del vector de suma de comprobación
1	OFF, número de pistas reservadas
2	PSH, desplazamiento de sector físico
3	PHM, máscara de sector físico
0	Una sola cara
40	Pistas por cara
9	Sectores por pista
1	Número del primer sector
512	Tamaño de sector
42	Longitud de intervalo (lectura/escritura)
82	Longitud de intervalo (formato)
60h (96)	Modo MFM, saltar marca de dirección de datos borrada
0	Auto-detectar formato del disco

Formato AMSTRAD CPC de 'sistema' (tipo 1)

36	SPT, sectores por pista
3	BSH, desplazamiento de bloque
7	BLM, máscara de bloque
0	EXM, máscara de sección
170	DSM, número de bloques − 1
63	DRM, número de reseñas del directorio − 1
C0h (192)	AL0, 2 bloques de directorio
00h (0)	AL1
16	CKS, tamaño del vector de suma de comprobación

2	OFF, número de pistas reservadas
2	PSH, desplazamiento de sector físico
3	PHM, máscara de sector físico
0	Una sola cara
40	Pistas por cara
9	Sectores por pista
41h (65)	Número del primer sector
512	Tamaño de sector
42	Longitud de intervalo (lectura/escritura)
82	Longitud de intervalo (formato)
60h (96)	Modo MFM, saltar marca de dirección de datos borrada
0	Auto-detectar formato del disco

Formato AMSTRAD CPC de 'datos' (tipo 2)

36	SPT, sectores por pista
3	BSH, desplazamiento de bloque
7	BLM, máscara de bloque
0	EXM, máscara de sección
179	DSM, número de bloques − 1
63	DRM, número de reseñas del directorio − 1
C0h (192)	AL0, 2 bloques de directorio
00h (0)	AL1
16	CKS, tamaño del vector de suma de comprobación
0	OFF, número de pistas reservadas
2	PSH, desplazamiento de sector físico
3	PHM, máscara de sector físico
0	Una sola cara
40	Pistas por cara
9	Sectores por pista
C1h (193)	Número del primer sector
512	Tamaño de sector
42	Longitud de intervalo (lectura/escritura)
82	Longitud de intervalo (formato)
60h (96)	Modo MFM, saltar marca de dirección de datos borrada
0	Auto-detectar formato del disco

Compatibilidad de ficheros de CP/M

+3DOS utiliza la estructura de ficheros de CP/M, con las siguientes limitaciones:

- Tamaño máximo de los ficheros: 8 megabytes (en CP/M Plus es 32 megabytes).
- Tamaño máximo de las unidades: 8 megabytes (en CP/M Plus es 128 megabytes).
- +3DOS ignora las etiquetas de directorio.
- +3DOS no admite protección con palabras clave. Los XFCB son borrados, etc. junto con sus ficheros, pero +3DOS los ignora para todo lo demás.
- +3DOS no admite estampación de fecha y hora. Los SFCB son inicializados a cero cuando se crea el fichero, pero +3DOS los ignora para todo lo demás.
- El atributo de 'archivo' es ignorado (o sea, la única rutina que le afecta es DOS EST ATRIB).

Estructura de los ficheros

Un fichero es una sucesión de bytes, cuya longitud puede ser de entre 0 y 8 megabytes. Cada fichero abierto tiene asociado un puntero de 24 bits. Este puntero da la dirección del próximo byte que el sistema va a leer o escribir. El puntero avanza automáticamente al siguiente byte tras cada operación de lectura o escritura; no obstante, el programa de usuario puede asignarle cualquier valor requerido para acceder al fichero en modo directo (aleatorio).

La posición de 'fin de fichero' (EOF) es la primera posición de byte que es posterior a todas las posiciones en las que se ha escrito algún byte. Los ficheros sin cabecera sólo pueden grabar su posición de EOF en el primer byte del siguiente registro de 128 bytes. Los ficheros con cabecera pueden tener su posición de EOF grabada en el sitio exacto.

Al escribir un byte en cualquier posición posterior a la de EOF se agranda el fichero y se hace avanzar la posición de EOF.

Si se intenta leer un byte en la posición de EOF o en cualquiera posterior se provoca un error.

Si se lee un byte en una posición anterior a la EOF pero en la que no se ha escrito previamente, se obtiene un valor sin significado, o bien se provoca un error. (No se recomienda leer posiciones en las que no se haya escrito previamente.)

Cambio de disco

En +3DOS se puede cambiar o extraer el disco en cualquier momento en que el sistema no esté accediendo a la unidad (y a condición de que no haya ningún fichero abierto). No es necesario, pues, hacer nada explícitamente para que el sistema 'reconozca' el disco.

No se debe cambiar el disco mientras alguno de sus ficheros esté abierto. No obstante, si se cambia el disco en esa situación, cuando +3DOS detecte este hecho pedirá al usuario que vuelva a introducir el disco correcto. (+3DOS sólo detectará el cambio de disco cuando vuelva a leer el directorio.)

Cambiar el disco cuando el sistema está leyendo o escribiendo en él puede ocasionar la destrucción de sus datos.

Proyección de unidades lógicas a unidades físicas

Si es necesario, se puede 'proyectar' dos unidades lógicas (A y B) hacia una sola unidad física (la 0). Este sistema puede interesar cuando sólo se ha conectado una unidad de disquete externa.

Para habilitar esta proyección se ejecuta la rutina DOS PROYEC B, a la que hay que proporcionar la dirección de la rutina CAMBIAR DISCO. Después de esto, +3DOS determina, cada vez que accede a la unidad física, si el disco es el correspondiente a la unidad lógica requerida. Si no lo es, invoca la rutina CAMBIAR DISCO; a ésta hay que proporcionarle la dirección de un mensaje y el nombre de la unidad lógica requerida. +3DOS emitirá un mensaje tal como:

Introduzca en la unidad el disco
para *x*: y luego pulse una tecla

donde *x* es el nombre de la unidad lógica (**A** o **B**). Esta rutina, antes de retornar, esperará hasta que el usuario pulse una tecla; entonces +3DOS supone que el disco que hay en la unidad 0 es el correspondiente a la unidad lógica especificada.

DOS PROYEC B sirve también para proyectar la unidad B hacia la 1. Si al ejecutar DOS PROYEC B de esta forma no existe la unidad física 1, la B quedará inhabilitada.

Códigos de error de +3DOS

Muchas de las rutinas de +3DOS pueden fracasar por diversas razones. A la salida de una rutina que ha fracasado, el indicador de arrastre queda puesto a 0 a la salida y el acumulador contiene el código del error.

Códigos de errores recuperables:

0	**UNIDAD NO PREPARADA**
1	**DISCO PROTEGIDO**
2	**FALLO DE BUSQUEDA**
3	**ERROR DE DATOS**
4	**SIN DATOS**
5	**SIN MARCA DIRECCIONES**
6	**FORMATO NO RECONOCIDO**
7	**ERROR DESCONOCIDO**
8	**DISCO CAMBIADO**
9	**SOPORTE NO ADECUADO**

Códigos de errores no recuperables:

20	**NOMBRE INCORRECTO**
21	**PARAMETRO INCORRECTO**
22	**UNIDAD NO ENCONTRADA**
23	**FICHERO NO ENCONTRADO**
24	**YA EXISTE EL FICHERO**
25	**FIN DE FICHERO**
26	**DISCO LLENO**
27	**DIRECTORIO LLENO**
28	**FICHERO SOLO LECTURA**
29	**FICHERO NO ABIERTO** (o abierto en un modo de acceso incorrecto)
30	**ACCESO DENEGADO** (cuando el fichero ya está siendo utilizado)
31	**SINTAXIS INCORRECTA**
32	**FALTA SECCION** (cuando no tendría que faltar)
33	**FALTA CACHE** (error de programa)
34	**FICH. DEMASIADO GRANDE** (se ha intentado leer o escribir más allá de los primeros 8 megabytes)
35	**DISCO NO DE ARRANQUE** (el sector cargador de inicialización no es aceptable para DOS CARGAR)
36	**UNIDAD YA EN USO** (se ha intentado reproyectar o deshabilitar una unidad en la que había ficheros abiertos)

Por ejemplo, el mensaje **SOPORTE NO ADECUADO** aparece cuando se intenta escribir sobre un disco de pista sencilla en una unidad de pista doble, o cuando se intenta leer o escribir sobre un disco de pista doble en una unidad de pista sencilla.

El mensaje **SIN MARCA DIRECCIONES** es el que aparece cuando se intenta usar un disco que no ha sido formateado (pero ésta no es la única causa del mensaje).

Mensajes de +3DOS

Si están habilitados los mensajes de error (DOS EST MENSAJE) y se produce un error recuperable, +3DOS entrega un mensaje a la rutina ALERTA y ésta pregunta al usuario: '**¿REINTENTAR, IGNORAR O CANCELAR?**'. Si la respuesta es '**R**', el sistema reintenta la operación en curso. Si es '**I**', el error es ignorado. Si es '**C**', el sistema abandona la operación y retorna con un código de error a la rutina original. Si los mensajes de error están inhibidos, o si el error que se produce es irrecuperable, el sistema no muestra ningún mensaje y retorna con un código de error a la rutina original.

Los errores de disco recuperables son:

0	**UNIDAD *x*: NO PREPARADA**
1	**UNIDAD *x*: DISCO PROTEGIDO CONTRA ESCRITURA**
2	**UNIDAD *x*: PISTA *ppp*, FALLO DE BUSQUEDA**
3	**UNIDAD *x*: PISTA *ppp*, SECTOR *sss*, ERROR DE DATOS**
4	**UNIDAD *x*: PISTA *ppp*, SECTOR *sss*, SIN DATOS**
5	**UNIDAD *x*: PISTA *ppp*, SECTOR *sss*, FALTA MARCA DE DIRECCIONES**
6	**UNIDAD *x*: FORMATO INCORRECTO**
7	**UNIDAD *x*: PISTA *ppp*, SECTOR *sss*, ERROR DESCONOCIDO**
8	**UNIDAD *x*: DISCO CAMBIADO; SUSTITUYALO**
9	**UNIDAD *x*: DISCO NO ADECUADO**

donde *x* representa la unidad de disco (**A** o **B**), *ppp* es el número de pista y *sss* es el número de sector.

Estos mensajes van seguidos de:

¿REINTENTAR, IGNORAR O CANCELAR?

+3DOS invoca la rutina ALERTA para emitir uno de estos mensajes si se produce un error cuando el sistema se ha puesto a ejecutar una rutina 'DOS'. Por ejemplo, si ejecutamos DOS ABRIR (en modo 'escritura exclusiva' o 'lectura/escritura exclusiva') y el disco está protegido contra escritura, DOS ABRIR retorna inmediatamente con el indicador de arrastre puesto a 0 y un código de error en el acumulador (sin invocar la rutina ALERTA).

Sin embargo, si ejecutamos la rutina DOS LEER para leer datos y ésta detecta un sector defectuoso, el sistema invoca la rutina ALERTA, la cual ofrece al usuario las opciones de reintentar (en caso, por ejemplo, de que el disco no esté bien introducido en la unidad),

ignorar (para que el sistema ignore el sector defectuoso y así poder recuperar el fichero en la mayor medida posible) o cancelar (cuando el problema sea claramente irrecuperable).

(Nótese que el interfaz para DOS EST MENSAJE ha cambiado de la versión V1.0 de +3DOS a la V1.1. Es importante invocar DOS VERSION para averiguar qué interfaz se debe usar. Ésta es la única diferencia entre las dos versiones de +3DOS y sólo afecta a las máquinas distribuidas fuera del Reino Unido.)

Requisitos de +3DOS

En el momento de invocar cualquier rutina de +3DOS es necesario que la configuración de la memoria sea la siguiente:

C000h...FFFFh	(49152...65535)	Página 7
8000h...BFFFh	(32768...49151)	Página 2
4000h...7FFFh	(16384...32767)	Página 5
0000h...3FFFh	(0...16383)	ROM 2

La pila tiene que estar por debajo de BFE0h (49120) y por encima de 4000h (16384). El límite superior es BFE0h (en lugar de C000h) porque los 30 últimos bytes de la página 2 están dedicados a implementar las traslaciones de bloques entre páginas. Este área está reservada para +3DOS; lo único que se requiere es que la pila no se encuentre en ella. La pila debe disponer de 50 palabras como mínimo.

+3DOS puede mantener abiertos simultáneamente hasta 16 ficheros. Los ficheros de número 0, 1 y 2 están reservados para +3 BASIC; por consiguiente, no se debe usar estos números si existe la posibilidad de que se ejecute una orden de +3 BASIC estando abierto uno de estos ficheros. El fichero 0 se cierra siempre que +3 BASIC emite un mensaje (aun cuando se trate de **'0 OK'**).

Para ejecutar las rutinas que vamos a describir, es necesario que las interrupciones estén habilitadas (y lo seguirán estando a la salida de la rutina).

Utilización de la memoria en +3DOS

Las páginas 1, 3, 4 y 6 de la RAM son tratadas como una sucesión de 128 tampones sectoriales (numerados del 0 al 127), de 512 bytes cada uno. El disco de RAM y el caché ocupan dos zonas (contiguas) de esta sucesión; su tamaño y posición son definidos en la inicialización, pero es posible redefinirlos más tarde. Los tampones no ocupados por el disco de RAM y el caché están disponibles para cualquier otro propósito. Al modificar el tamaño o la posición del disco de RAM se borra todos sus ficheros.

Todas las rutinas de +3DOS retornan con la misma configuración de memoria que había a la entrada.

Las direcciones de nombres de fichero, tampones, etc. entregadas a las rutinas tienen que ser 'visibles'; o sea, la página de RAM en la que se encuentren tiene que estar encajada en la memoria accesible.

El bloque de saltos de DOS está en la ROM 2, a partir de la dirección 0100h (256). Las direcciones y los interfaces de las rutinas son los detallados en los apartados siguientes.

Rutinas esenciales del sistema de archivo

DOS INICIALIZAR
0100h (256)

- Inicializar +3DOS.
- Inicializar los controladores de disco.
- Inicializar el caché y el disco de RAM.
- Todos los ficheros cerrados.
- Todos los discos 'no reconocidos'.
- Unidad por defecto: si está presente el interfaz del disco, la A; si no, la M.
- Número de usuario por defecto: 0.
- Contador de 'reintentar' puesto a 15.
- Mensajes de error inhibidos.

Condiciones de entrada

Ninguna

Situación de salida

Éxito:
Indicador de arrastre a 1
A corrupto

Fracaso:
Indicador de arrastre a 0
A = código del error

Siempre:
BC, DE, HL, IX corruptos
Preservados todos los demás registros

DOS VERSION
0103h (259)

• Leer los números de edición y versión de DOS.

Condiciones de entrada

Ninguna

Situación de salida

D = edición
E = versión (dentro de la edición)

Siempre:

AF, BC, HL, IX corruptos
Preservados todos los demás registros

DOS ABRIR
0106h (262)

• Crear y/o abrir un fichero.

Si ya existe el fichero, la rutina ejecuta la 'acción abrir'; si no, ejecuta la 'acción crear'. Estas funciones son especificadas en DE, con los siguientes valores:

Acción abrir:

1 = Error — Ya existe el fichero.
2 = Abrir el fichero, leer la cabecera (si la hay). Colocar el puntero inmediatamente después de la cabecera.
3 = Abrir el fichero, ignorar la cabecera. Colocar el puntero en 0000h (0).
4 = Suponiendo que el nombre de fichero especificado es *nombre.tipo*, borrar *nombre*.**BAK** (si existe); a *nombre.tipo* darle el nombre *nombre*.**BAK**. Ejecutar la 'acción crear'.
5 = Borrar la versión existente. Ejecutar la 'acción crear'.

Acción crear:

1 = Error — No existe el fichero.
2 = Crear y abrir el fichero nuevo con cabecera. Colocar el puntero inmediatamente después de la cabecera.
3 = Crear y abrir el fichero nuevo sin cabecera. Colocar el puntero en 000000h (0).

Ejemplo. Para simular en la **cinta** la acción 'si el fichero existe, abrirlo; si no, crearlo con cabecera', se debe hacer 'acción abrir' igual a 1 y 'acción crear' igual a 1.

Ejemplo. Para abrir un fichero y generar un mensaje de error en caso de que el fichero no exista, se debe hacer 'acción abrir' igual a 1 y 'acción crear' igual a 0.

Ejemplo. Para crear un fichero nuevo con cabecera, cambiando previamente la versión existente a **.BAK**, se debe hacer 'acción abrir' igual a 3 y 'acción crear' igual a 1.

Para los ficheros **con** cabecera la posición de EOF es la primera posición de byte que es posterior a todas las posiciones en las que se ha escrito algún byte.

Para los ficheros **sin** cabecera la posición de EOF es el primer byte del primer registro de 128 bytes que es posterior a todas las posiciones en las que se ha escrito algún byte.

El EOF 'blando' (fin de fichero por programa) es el carácter 1Ah (26) y no tiene nada que ver con la posición de EOF. Únicamente DOS LEER BYTE reconoce el EOF blando.

El área de datos de cabecera consta de 8 bytes; el programa invocante puede usarlos con cualquier propósito. Si la 'acción abrir' es 1 y el fichero existe (y tiene cabecera), el sistema lee los datos de cabecera en el fichero; en cualquier otro caso, pone a cero todos los datos de cabecera.

Los datos de cabecera están disponibles aun cuando el fichero no tenga cabecera. Para acceder a ellos se ejecuta DOS REF CAB.

+3 BASIC utiliza los 7 primeros de esos 8 bytes de la siguiente manera:

Byte	0	1	2	3	4	5	6
Programa	0	Longitud de fichero		8000h o **LINE**		Distancia hasta prog	
Matriz numérica	1	Longitud de fichero		—	nombre	—	—
Matriz literal	2	Longitud de fichero		—	nombre	—	—
CODE o **SCREEN$**	3	Longitud de fichero		Dirección de carga		—	—

('—' significa 'indiferente'.)

Al crear un fichero que luego vaya a ser cargado por BASIC (con **LOAD**) estos bytes deben ser rellenados con los valores apropiados.

Si se abre el fichero en modo de 'escritura exclusiva' o de 'lectura/escritura exclusiva' (y si el fichero tiene cabecera), el sistema actualiza la cabecera al cerrarlo.

Si un fichero está abierto en modo de 'lectura compartida' con cierto número, sólo es posible abrirlo en el mismo modo con otro número.

Si un fichero está abierto en modo de 'escritura exclusiva' o de 'lectura/escritura exclusiva' con cierto número, no es posible abrirlo con otro número.

Condiciones de entrada

B = número de fichero, 0...15
C = modo de acceso requerido
valor en los bits 0...2:
1 = lectura exclusiva
2 = escritura exclusiva
3 = lectura/escritura exclusiva
5 = lectura compartida
bits 3...7 = 0 (reservados)
D = acción crear
E = acción abrir
HL = dirección del nombre de fichero (sin caracteres polivalentes)

Situación de salida

Si se ha creado un fichero nuevo:
Indicador de arrastre a 1
Indicador de cero a 1
A corrupto

Si se ha abierto un fichero existente:
Indicador de arrastre a 1
Indicador de cero a 0
A corrupto

En cualquier otro caso:
Indicador de arrastre a 0
A = código del error

Siempre:
BC, DE, HL, IX corruptos
Preservados todos los demás registros

DOS CERRAR
0109h (265)

- Cerrar un fichero.
- Escribir la cabecera (si existe).
- Escribir los datos pendientes.
- Actualizar el directorio.
- Liberar el número de fichero.

Todos los ficheros abiertos deben ser cerrados (o abandonados) en algún momento. El número con el que se abre un fichero no puede volver a ser utilizado mientras no se cierre el fichero.

Condiciones de entrada

B = número del fichero

Situación de salida

Éxito:
- Indicador de arrastre a 1
- A corrupto

Fracaso:
- Indicador de arrastre a 0
- A = código del error

Siempre:
- BC, DE, HL, IX corruptos
- Preservados todos los demás registros

DOS ABANDONAR
010Ch (268)

• Abandonar un fichero.

El efecto de esta rutina es similar al de DOS CERRAR, con la diferencia de que no escribe en el disco la cabecera ni los datos pendientes y no actualiza el directorio. Sólo se debe usar esta rutina en caso de que DOS CERRAR no sea capaz de cerrar el fichero (por ejemplo, cuando el disco es defectuoso o ha sido extraído de la unidad o cambiado permanentemente).

Condiciones de entrada

B = número del fichero

Situación de salida

Éxito:
- Indicador de arrastre a 1
- A corrupto

Fracaso:
- Indicador de arrastre a 0
- A = código del error

Siempre:
- BC, DE, HL, IX corruptos
- Preservados todos los demás registros

DOS REF CAB
010Fh (271)

• Apuntar a los datos de cabecera del fichero.

El área de datos de cabecera consta de 8 bytes, que pueden ser utilizados con cualquier propósito. Los datos de cabecera están disponibles aun cuando el fichero no tenga cabecera. Sin embargo, sólo serán grabados en el disco si el fichero tiene cabecera y está abierto para escritura.

+3 BASIC utiliza estos bytes de la forma descrita en DOS ABRIR. Al crear un fichero que luego vaya a ser cargado por BASIC (con **LOAD**) estos bytes deben ser rellenados con los valores apropiados.

Condiciones de entrada

B = número del fichero

Situación de salida

Éxito, fichero sin cabecera:
Indicador de arrastre a 1
Indicador de cero a 1
A corrupto
IX = dirección de los datos de cabecera (en la página 7)

Éxito, fichero con cabecera:
Indicador de arrastre a 1
Indicador de cero a 0
A corrupto
IX = dirección de los datos de cabecera (en la página 7)

Fracaso:
Indicador de arrastre a 0
A = código del error
IX = dirección de los datos de cabecera (en la página 7)

Siempre:
BC, DE, HL corruptos
Preservados todos los demás registros

DOS LEER
0112h (274)

• Leer bytes en un fichero y copiarlos en la memoria.
• Hacer avanzar el puntero del fichero.

El tampón de destino tiene la siguiente configuración:

C000h...FFFFh	(49152...65535)	Página especificada en C
8000h...BFFFh	(32768...49151)	Página 2
4000h...7FFFh	(16384...32767)	Página 5
0000h...3FFFh	(0...16383)	ROM de DOS

Esta rutina ignora los EOF blandos.

Se produce un error si se intenta leer en la posición de EOF o más allá.

Condiciones de entrada

B = número del fichero
C = página que entra en C000h...FFFFh (49152...65535)
DE = número de bytes que deben ser leídos (0 significa 64K)
HL = dirección de los bytes que deben ser leídos

Situación de salida

Éxito:
Indicador de arrastre a 1
A y DE corruptos

Fracaso:
Indicador de arrastre a 0
A = código del error
DE = número de bytes que quedan por leer

Siempre:
BC, HL, IX corruptos
Preservados todos los demás registros

DOS ESCRIBIR
0115h (277)

- Escribir bytes en un fichero copiándolos desde la memoria.
- Hacer avanzar el puntero del fichero.

El tampón de destino tiene la siguiente configuración:

C000h...FFFFh	(49152...65535)	Página especificada en C
8000h...BFFFh	(32768...49151)	Página 2
4000h...7FFFh	(16384...32767)	Página 5
0000h...3FFFh	(0...16383)	ROM de DOS

Condiciones de entrada

B = número del fichero
C = página que entra en C000h...FFFFh (49152...65535)
DE = número de bytes que deben ser escritos (0 significa 64K)
HL = dirección de los bytes que deben ser escritos

Situación de salida

Éxito:
Indicador de arrastre a 1
A y DE corruptos
Fracaso:
Indicador de arrastre a 0
A = código del error
DE = número de bytes que quedan por escribir
Siempre:
BC, HL, IX corruptos
Preservados todos los demás registros

DOS LEER BYTE
0118h (280)

- Leer un byte en un fichero.
- Hacer avanzar el puntero del fichero.

Esta rutina comprueba si el byte leído es un EOF blando, carácter 1Ah (26). Es posible leer más allá de los EOF blandos.

El programa invocante debe decidir si le afecta la detección de un EOF blando (normalmente sólo lo hará cuando se está leyendo un fichero ASCII).

Se produce un error si se intenta leer en la posición de EOF o más allá.

Condiciones de entrada

B = número del fichero

Situación de salida

Éxito, el byte leído no es EOF blando (1Ah):
Indicador de arrastre a 1
Indicador de cero a 0
A corrupto
C = byte leído

Éxito, el byte leído es EOF blando (1Ah):
Indicador de arrastre a 1
Indicador de cero a 1
A corrupto
C = byte leído

En cualquier otro caso:
Indicador de arrastre a 0
A = código del error
C corrupto

Siempre:
B, DE, HL, IX corruptos
Preservados todos los demás registros

DOS ESCR BYTE
011Bh (283)

- Escribir un byte en un fichero.
- Hacer avanzar el puntero del fichero.

Condiciones de entrada

B = número del fichero
C = byte que debe ser escrito

Situación de salida

Éxito:
Indicador de arrastre a 1
A corrupto

Fracaso:
Indicador de arrastre a 0
A = código del error

Siempre:
BC, DE, HL, IX corruptos
Preservados todos los demás registros

DOS CATALOGO
011Eh (286)

- Llenar un tampón con parte del directorio (en orden alfanumérico).

La especificación de fichero incluye la letra de unidad, el número de usuario y el nombre de fichero (posiblemente ambiguo, es decir, construido con caracteres polivalentes).

Puesto que el tamaño del directorio es variable (y puede ser bastante grande), esta rutina permite que el directorio sea catalogado en pequeñas secciones. El programa invocante entrega un tampón precargado con el primer nombre de fichero deseado (o con ceros para representar el principio del directorio). La rutina deposita en el tampón parte del directorio (o el directorio completo, si cabe), en orden ASCII. Si no ha cabido el directorio solicitado, se reinvoca la rutina (con el tampón precargado con el último fichero entregado la vez anterior). El proceso se repite hasta que todo el directorio haya quedado catalogado.

Los discos de formato +3DOS (que tienen la misma estructura que los de una sola cara y pista sencilla del AMSTRAD PCW) pueden tener como máximo 64 reseñas en el directorio.

El formato del tampón es:

Reseña 0
Reseña 1
Reseña 2
Reseña 3
:
Reseña *n*

La reseña 0 tiene que ser precargada con el primer *nombre.tipo* requerido (o con ceros, si se desea el directorio completo). La reseña 1 contendrá el primer nombre de fichero que encaje en la especificación y sea mayor que el precargado.

Si el tampón no es suficientemente grande como para recibir el directorio completo, se puede volver a invocar la rutina (precargando la reseña 0 con el contenido de la reseña *n* de la vez anterior).

Formato de las reseñas (longitud de 13 bytes):

Bytes 0...7	Nombre del fichero (ASCII), alineado a la izquierda y rellenado con espacios por la derecha
Bytes 8...10	Tipo del fichero (ASCII), alineado a la izquierda y rellenado con espacios por la derecha
Bytes 11...12	Tamaño en kilobytes (binario)

El tamaño entregado por esta rutina es el espacio de disco asignado al fichero, que no necesariamente coincide con el tamaño real del fichero.

Condiciones de entrada

B = $n+1$, tamaño del tampón expresado en número de reseñas; >=2
C = filtro
bit 0 = incluir ficheros de sistema si este bit está a 1
bits 1...7 = 0 (reservados)
DE = dirección del tampón (la primera reseña tiene que estar inicializada)
HL = dirección del nombre de fichero (caracteres polivalentes permitidos)

Situación de salida

Éxito:
Indicador de arrastre a 1
A corrupto
B = número de reseñas depositadas en el tampón, 0...*n*. (Si es *n*, pueden haber quedado reseñas pendientes de catalogar.)

Fracaso:
Indicador de arrastre a 0
A = código del error
B corrupto

Siempre:
C, DE, HL, IX corruptos
Preservados todos los demás registros

DOS ESPACIO
0121h (289)

- Averiguar cuánto espacio queda libre en el disco.

Condiciones de entrada

A = letra de la unidad (ASCII 'A', ..., 'P')

Situación de salida

Éxito:
Indicador de arrastre a 1
A corrupto
HL = espacio libre en kilobytes

Fracaso:
Indicador de arrastre a 0
A = código del error
HL corrupto

Siempre:
BC, DE, IX corruptos
Preservados todos los demás registros

DOS BORRAR
0124h (292)

• Borrar un fichero.

El fichero no puede estar abierto.

Condiciones de entrada

HL = dirección del nombre de fichero (caracteres polivalentes permitidos)

Situación de salida

Éxito:
Indicador de arrastre a 1
A corrupto

Fracaso:
Indicador de arrastre a 0
A = código del error

Siempre:
BC, DE, HL, IX corruptos
Preservados todos los demás registros

DOS CAMB NOMBRE
0127h (295)

• Cambiar el nombre de un fichero.

El fichero no puede estar abierto. No puede existir un fichero que tenga el nombre nuevo. El nombre nuevo tiene que especificar, explícita o implícitamente, la misma unidad que el antiguo.

Condiciones de entrada

DE = dirección del nombre nuevo (sin caracteres polivalentes)
HL = dirección del nombre antiguo (sin caracteres polivalentes)

Situación de salida

Éxito:
 Indicador de arrastre a 1
 A corrupto

Fracaso:
 Indicador de arrastre a 0
 A = código del error

Siempre:
 BC, DE, HL, IX corruptos
 Preservados todos los demás registros

DOS CARGAR
012Ah (298)

- Cargar a partir del disco.

Esta rutina lee en el disco de la unidad A (si está conectada) un sector de inicialización, lo carga en la memoria e inicia su ejecución. El sector de inicialización cargará y ejecutará a su vez un programa de juego o un sistema operativo.

El entorno de memoria para el sector de inicialización es:

C000h...FFFFh	(49152...65535)	Página 3
8000h...BFFFh	(32768...49151)	Página 6
4000h...7FFFh	(16384...32767)	Página 7
0000h...3FFFh	(0...16383)	Página 4

El sector de inicialización debe estar en la cara 0, pista 0, sector 1. Esta rutina lo carga a partir de FE00h (65024) y lo ejecuta a partir de FE10h (65040). Las interrupciones están inhibidas. El puntero de pila está en FE00h (65024). La suma de todos los bytes del sector, tomada módulo 256, tiene que ser igual a 3 (se puede usar el byte 15 del sector para ajustar la suma al valor requerido).

Los bytes 0...15 del sector contienen la especificación del disco.

Condiciones de entrada

Ninguna

Situación de salida

Éxito:
 No hay salida, ya que el sector de inicialización toma el control de la máquina

Fracaso:
Indicador de arrastre a 0
A = código del error

Siempre:
BC, DE, HL, IX corruptos
Preservados todos los demás registros

DOS EST UNIDAD
012Dh (301)

- Seleccionar la unidad implícita (es decir, la unidad que el sistema toma por defecto cuando no se menciona ninguna en las especificaciones de fichero).
- Averiguar cuál es actualmente la unidad implícita.

Inicialmente la unidad implícita es la M (o la A, si está conectada la unidad externa).

Esta rutina no accede a la unidad, sino que se limita a comprobar que existe el controlador (lo cual no implica que exista la unidad).

La selección de unidad implícita sólo afecta a las rutinas que toman nombres de fichero como parámetros.

Condiciones de entrada

A = letra de la unidad (ASCII 'A', . . . , 'P'), [FFh (255) para averiguar cuál es la unidad implícita]

Situación de salida

Éxito:
Indicador de arrastre a 1
A = letra de la unidad

Fracaso:
Indicador de arrastre a 0
A = código del error

Siempre:
BC, DE, HL, IX corruptos
Preservados todos los demás registros

DOS EST USUARIO
0130h (304)

- Seleccionar el número de usuario implícito (es decir, el área de usuario que el sistema toma por defecto cuando no se menciona ninguna en las especificaciones de fichero).
- Averiguar cuál es actualmente el número de usuario implícito.

Inicialmente el número de usuario implícito es el 0.

La selección de número de usuario implícito sólo afecta a las rutinas que toman nombres de fichero como parámetros.

Condiciones de entrada

A = número de usuario (0. . .15), [FFh (255) para averiguar cuál es la unidad implícita]

Situación de salida

Éxito:
Indicador de arrastre a 1
A = número de usuario

Fracaso:
Indicador de arrastre a 0
A = código del error

Siempre:
BC, DE, HL, IX corruptos
Preservados todos los demás registros

Rutinas adicionales para juegos y sistemas operativos

DOS LEER POS
0133h (307)

- Averiguar la posición del puntero del fichero.

Condiciones de entrada

B = Número del fichero

Situación de salida

Éxito:
Indicador de arrastre a 1
A corrupto
E HL = puntero del fichero 000000h...FFFFFFh (0...16777215)
(E contiene el byte más significativo; L, el menos significativo)

Fracaso:
Indicador de arrastre a 0
A = código del error
E HL corruptos

Siempre:
BC, D, IX corruptos
Preservados todos los demás registros

DOS EST POS
0136h (310)

- Establecer la posición del puntero del fichero.

Esta rutina no accede a la unidad de disco. No comprueba si el puntero está más allá del límite de 8 megabytes ni le afecta este hecho.

Condiciones de entrada

B = Número del fichero
E HL = puntero del fichero 000000h...FFFFFFh (0...16777215)
(E contiene el byte más significativo; L, el menos significativo)

Situación de salida

Éxito:
Indicador de arrastre a 1
A corrupto

Fracaso:
Indicador de arrastre a 0
A = código del error

Siempre:
BC, DE, HL, IX corruptos
Preservados todos los demás registros

DOS LEER EOF
0139h (313)

- Averiguar la posición del final del fichero (EOF); es decir, la primera posición de byte que es posterior a todas las posiciones en las que se ha escrito algún byte.

Esta rutina no afecta al puntero del fichero e ignora los EOF blandos.

Condiciones de entrada

B = Número del fichero

Situación de salida

Éxito:
Indicador de arrastre a 1
A corrupto
E HL = puntero del fichero 000000h...FFFFFFh (0...16777215)
(E contiene el byte más significativo; L, el menos significativo)

Fracaso:
Indicador de arrastre a 0
A = código del error
E HL corruptos

Siempre:
BC, D, IX corruptos
Preservados todos los demás registros

DOS LEER 1346
013Ch (316)

- Averiguar la posición actual del caché y el disco de RAM.

Las páginas 1, 3, 4 y 6 son tratadas como una sucesión de 128 tampones sectoriales (numerados del 0 al 127), de 512 bytes cada uno. El disco de RAM y el caché ocupan dos zonas (contiguas) de esta sucesión.

El programa invocante puede usar cualquier sector que no esté ocupado.

Nótese que los tamaños pueden ser menores que los especificados en DOS EST 1346, ya que existe un valor máximo para el tamaño del caché y un valor mínimo para el tamaño del disco de RAM (4 sectores).

Condiciones de entrada

Ninguna

Situación de salida

D = primer tampón del caché
E = número de tampones dedicados al caché
H = primer tampón del disco de RAM
L = número de tampones dedicados al disco de RAM

Siempre:
AF, BC, IX corruptos
Preservados todos los demás registros

DOS EST 1346
013Fh (319)

• Reconstruir el caché y el disco de RAM.

Esta rutina libera espacio de almacenamiento para el programa de usuario, o lo reduce para asignárselo a DOS.

Al modificar el tamaño o la posición del disco de RAM se borra todos sus ficheros.

Las páginas 1, 3, 4 y 6 son tratadas como una sucesión de 128 tampones sectoriales (numerados del 0 al 127), de 512 bytes cada uno. El disco de RAM y el caché ocupan dos zonas (contiguas) de esta sucesión.

La posición y el tamaño del caché y del disco de RAM pueden ser especificados por separado. Los tampones no usados por éstos quedan disponibles para el programa de usuario, ya que DOS no los utiliza.

A pesar de que esta rutina no lo comprueba, el caché y el disco de RAM no deben quedar solapados.

Nótese que los tamaños reales pueden ser menores que los especificados, ya que existe un valor máximo para el tamaño del caché y un valor mínimo para el tamaño del disco de RAM (4 sectores).

Un caché de tamaño nulo es válido, pero dificultará en grado sumo el funcionamiento de la unidad de disco.

Esta rutina fracasa si está abierto alguno de los ficheros de la unidad M.

Condiciones de entrada

D = primer tampón del caché
E = número de tampones dedicados al caché
H = primer tampón del disco de RAM
L = número de tampones dedicados al disco de RAM
(Nótese que E + L <= 128)

Situación de salida

Éxito:
  Indicador de arrastre a 1
  A corrupto

Fracaso:
  Indicador de arrastre a 0
  A = código del error

Siempre:
  BC, DE, HL, IX corruptos
  Preservados todos los demás registros

DOS ACTUALIZAR
0142h (322)

- Escribir las cabeceras, los datos y las reseñas de directorio que estén pendientes de actualizar en esta unidad.

Esta rutina deja el disco totalmente actualizado. Puede ser ejecutada en cualquier momento, aun cuando haya ficheros abiertos.

Condiciones de entrada

A = letra de la unidad (ASCII 'A', . . ., 'P')

Situación de salida

Éxito:
  Indicador de arrastre a 1
  A corrupto

Fracaso:
  Indicador de arrastre a 0
  A = código del error

Siempre:
  BC, DE, HL, IX corruptos
  Preservados todos los demás registros

DOS EST ACCESO
0145h (325)

- Intentar cambiar el modo de acceso a un fichero que ya está abierto.

Esta rutina fracasa si el fichero está abierto en un modo de acceso incompatible, o si se solicita el acceso en escritura a un fichero o disco que es de 'sólo lectura'.

Condiciones de entrada

B = número de fichero
C = modo de acceso requerido
  valor en los bits 0...2:
    1 = lectura exclusiva
    2 = escritura exclusiva
    3 = lectura/escritura exclusiva
    5 = lectura compartida
    (todos los demás valores están reservados)
  bits 3...7 = 0 (reservados)

Situación de salida

Éxito:
  Indicador de arrastre a 1
  A corrupto

Fracaso:
  Indicador de arrastre a 0
  A = código del error

Siempre:
  BC, DE, HL, IX corruptos
  Preservados todos los demás registros

DOS EST ATRIB
0148h (328)

- Establecer los atributos de un fichero.

Sólo se puede modificar los atributos f1'...f4' y t1'...t4'. Los restantes, f5'...f8', están siempre a cero.

Esta rutina primero activa los atributos especificados en D y luego cancela los especificados en E. O sea, E tiene prioridad sobre D.

Condiciones de entrada

D = atributos que se desea activar:
bit 0 = t3' archivo
bit 1 = t2' sistema
bit 2 = t1' sólo lectura
bit 3 = f4'
bit 4 = f3'
bit 5 = f2'
bit 6 = f1'

E = atributos que se desea cancelar:
bit 0 = t3' archivo
bit 1 = t2' sistema
bit 2 = t1' sólo lectura
bit 3 = f4'
bit 4 = f3'
bit 5 = f2'
bit 6 = f1'

HL = dirección del nombre de fichero (caracteres polivalentes permitidos)

Situación de salida

Éxito:
Indicador de arrastre a 1
A corrupto

Fracaso:
Indicador de arrastre a 0
A = código del error

Siempre:
BC, DE, HL, IX corruptos
Preservados todos los demás registros

DOS ABRIR UNIDAD
014Bh (331)

- Abrir el disco de la unidad como un único fichero.

El disco entero queda accesible, cualquiera que sea su estructura de ficheros. Esta rutina puede ser utilizada para examinar y modificar directorios, ficheros, etc., pero hay que hacerlo con gran precaución, pues se corre el riesgo de perder los ficheros.

Esta rutina coloca el puntero del fichero en 000000h (0).

Si ya hay algún fichero abierto en modo de lectura compartida, el disco sólo puede ser abierto en ese mismo modo.

Si ya hay algún fichero abierto en modo de acceso exclusivo, no se puede abrir el disco.

Condiciones de entrada

A = letra de unidad (ASCII 'A'...'P')
B = número de fichero
C = modo de acceso requerido
valor en los bits 0...2:
1 = lectura exclusiva
2 = escritura exclusiva
3 = lectura/escritura exclusiva
5 = lectura compartida
(todos los demás valores están reservados)
bits 3...7 = 0 (reservados)

Situación de salida

Éxito:
Indicador de arrastre a 1
A corrupto
Fracaso:
Indicador de arrastre a 0
A = código del error
Siempre:
BC, DE, HL corruptos
Preservados todos los demás registros

DOS EST MENSAJE
014Eh (334)

- Habilitar o inhibir los mensajes de error.

Se debe ejecutar esta rutina para informar a +3DOS de que existe una subrutina 'ALERTA' de usuario. Cuando +3DOS detecta un error, invoca la subrutina ALERTA y le entrega los valores detallados más abajo. ALERTA debe exhibir el texto del mensaje que +3DOS le ha entregado y luego esperar hasta que el usuario pulse una tecla. Si el carácter generado por la tecla está en la cadena de respuesta (que +3DOS también ha entregado a ALERTA, aunque sólo en la versión V1.0), ALERTA debe retornar con 0, 1, 2 o el carácter en el acumulador (dependiendo de la versión de +3DOS).

Condiciones de entrada

A = habilitar/inhibir mensajes
FFh (255) = habilitar
00h (0) = inhibir
HL = dirección de la rutina ALERTA (en caso de habilitar)

Situación de salida

HL = dirección de la ALERTA anterior (0 si no existía)

Siempre:
AF, BC, DE, IX corruptos
Preservados todos los demás registros

Nota

En caso de existir una subrutina ALERTA de usuario, las 'condiciones de entrada' son las condiciones en que se debe entrar en ella, mientras que la 'situación de salida' indica qué valores debe generar la subrutina y qué registros le está permitido corromper.

Hay dos interfaces para ALERTA, que dependen de la versión de +3DOS:

ALERTA (sólo versión V1.0)

Condiciones de entrada

DE = dirección de la cadena de respuesta (en la página 7), terminada en FFh (255)
HL = dirección del mensaje de error (en la página 7), terminado en FFh (255)

Situación de salida

A = carácter con que el usuario ha respondido

Siempre:
F, BC, DE, HL, IX corruptos
Preservados todos los demás registros

La segunda versión de ALERTA que permite que el programa de usuario ofrezca mensajes de error distintos de los británicos está disponible en las versiones V1.1 y posteriores.

ALERTA (Versiones V1.0 y posteriores)

Condiciones de entrada

B = número del error
C = letra de unidad (ASCII 'A'...'P')
D = pista lógica (si el mensaje la necesita)
E = sector lógico (si el mensaje lo necesita)
HL = dirección del mensaje de error británico (en la página 7), terminado en FFh (255)

Situación de salida

A = respuesta
0 = cancelar
1 = reintentar
2 = ignorar

Si el programa de usuario proporciona su propia función de ALERTA, tiene que disponer de dos subrutinas (o de una con condiciones de entrada y de salida conmutables) y averiguar cuál es la versión de +3DOS antes de elegir una de ellas.

DOS REF XDPB
0151h (337)

- Apuntar al XDPB de esta unidad. (El XDPB, bloque de parámetros de disco ampliado, es utilizado por las rutinas del controlador de disco.)

Condiciones de entrada

A = letra de unidad (ASCII 'A'...'P')

Situación de salida

Éxito:
Indicador de arrastre a 1
A corrupto
IX = dirección del XDPB

Fracaso:
Indicador de arrastre a 0
A = código del error
IX corrupto

Siempre:
BC, DE, HL corruptos
Preservados todos los demás registros

DOS PROYEC B
0154h (340)

- Proyectar la unidad B hacia la 0 o la 1.

Esta rutina fracasa si está abierto alguno de los ficheros de la unidad B.

Si se proyecta la unidad B hacia la 0, el sistema comprobará en lo sucesivo que el disco es el correcto cada vez que acceda a la unidad 0. Si no lo es, invocará la subrutina CAMBIAR DISCO para pedir al usuario que cambie el disco.

Si se proyecta la unidad B hacia la 1 y ésta no existe, la unidad B queda inhabilitada.

Condiciones de entrada

C = unidad (0 o 1)
HL = dirección de la subrutina CAMBIAR DISCO si la unidad es la 0

Situación de salida

Éxito:
Indicador de arrastre a 1
A corrupto
HL = dirección de la anterior subrutina CAMBIAR DISCO (0 si no existía)

Fracaso:
Indicador de arrastre a 0
A HL corruptos

Siempre:
BC, DE, IX corruptos
Preservados todos los demás registros

Nota

En caso de existir una subrutina CAMBIAR DISCO de usuario, las 'condiciones de entrada' son las condiciones en que se debe entrar en ella, mientras que la 'situación de salida' indica qué registros le está permitido corromper.

CAMBIAR DISCO

- Pedir al usuario que cambie el disco de la unidad 0.
- Esperar hasta que el usuario avise de que ha realizado el cambio.

Condiciones de entrada

A = unidad lógica (ASCII 'A'...'P')
HL = dirección del mensaje (en la página 7), terminado en FFh (255)

Situación de salida

Siempre:
AF, BC, DE, HL, IX corruptos
Preservados todos los demás registros

Rutinas de control del disco a bajo nivel

A continuación vamos a describir las rutinas del controlador de la unidad de disquete. El número de unidad física está entre 0 y 3 para el μPD765A. En el **+2A** la unidad lógica

A es la unidad física 0 y la unidad lógica B es la unidad física 1, si bien la B puede ser proyectada hacia la 0. El **+2A** no utiliza las unidades 2 y 3.

Con la excepción de DD INTERFAZ, no se debe invocar ninguna de estas rutinas si no está presente el interfaz de la unidad de disco.

Todas las rutinas suponen que las interrupciones están habilitadas a la entrada, y lo seguirán estando a la salida.

DD INTERFAZ
0157h (343)

• Averiguar si está presente el interfaz controlador de la unidad de disquete.

(Esta información se encuentra también en el bit 4 de la variable de sistema FLAGS3 de BASIC.)

Condiciones de entrada

Ninguna

Situación de salida

Si está presente:
Indicador de arrastre a 1

Si no lo está:
Indicador de arrastre a 0

Siempre:
A, BC, DE, HL, IX corruptos
Preservados todos los demás registros

DD INIC
015Ah (346)

• Inicializar el controlador de la unidad de disco.

Condiciones de entrada

Ninguna

Situación de salida

Siempre:
AF, BC, DE, HL, IX corruptos
Preservados todos los demás registros

DD CONFIGURAR
015Dh (349)

• Especificar los parámetros de la unidad de disco.

Formato del bloque de parámetros:

Byte 0 — Tiempo de espera después de la puesta en marcha del motor (en unidades de 100 ms)
Byte 1 — Tiempo de espera después de la parada del motor (en unidades de 100 ms)
Byte 2 — Tiempo de espera después de escritura (en unidades de 10 μs)
Byte 3 — Tiempo de estabilización de la cabeza (en unidades de 1 ms)
Byte 4 — Tiempo de avance de pasos (en unidades de 1 ms)
Byte 5 — Tiempo de descarga de la cabeza (en unidades de 32 ms, 32...480)
Byte 6 — (Tiempo de carga de la cabeza × 2) + 1 (en unidades de 4 ms, 4...508)

Condiciones de entrada

HL = dirección del bloque de parámetros

Situación de salida

Siempre:
AF, BC, DE, HL, IX corruptos
Preservados todos los demás registros

DD EST REINT
0160h (352)

• Ajustar el contador de 'reintentar'.

(Si se especifica el valor 1, el sistema sólo intenta la operación una vez, es decir, no hay reintentos.)

Condiciones de entrada

A = valor del contador, >=1

Situación de salida

Siempre:
AF, BC, DE, HL, IX corruptos
Preservados todos los demás registros

DD LEER SECTOR
0163h (355)

• Leer un sector.

Condiciones de entrada

B = página que entra en C000h...FFFFh (49152...65535)
C = unidad (0 o 1)
D = pista lógica (base 0)
E = sector lógico (base 0)
HL = dirección del tampón
IX = dirección del XDPB

Situación de salida

Éxito:
Indicador de arrastre a 1
A corrupto
Fracaso:
Indicador de arrastre a 0
A = código del error
Siempre:
BC, DE, HL, IX corruptos
Preservados todos los demás registros

DD ESCR SECTOR
0166h (358)

• Escribir un sector.

Condiciones de entrada

B = página que entra en C000h...FFFFh (49152...65535)
C = unidad (0 o 1)
D = pista lógica (base 0)
E = sector lógico (base 0)
HL = dirección del tampón
IX = dirección del XDPB

Situación de salida

Éxito:
Indicador de arrastre a 1
A corrupto

Fracaso:
Indicador de arrastre a 0
A = código del error

Siempre:
BC, DE, HL, IX corruptos
Preservados todos los demás registros

DD VERIF SECTOR
0169h (361)

- Verificar un sector. (Utiliza la orden de verificación 'igual' del μPD765A.)

Esta rutina comprueba que el sector del disco es idéntico a la copia almacenada en la memoria.

El byte FFh (255) en el disco (o en la memoria) concuerda con cualquier valor que la rutina encuentre en la memoria (o en el disco). (Véase la especificación del μPD765A para más detalles.)

Condiciones de entrada

B = página que entra en C000h...FFFFh (49152...65535)
C = unidad (0 o 1)
D = pista lógica (base 0)
E = sector lógico (base 0)
HL = dirección de la copia del sector
IX = dirección del XDPB

Situación de salida

Éxito, el sector es igual a la copia:
Indicador de arrastre a 1
Indicador de cero a 1
A corrupto

Éxito, el sector es distinto de la copia:
Indicador de arrastre a 1
Indicador de cero a 0
A corrupto

En cualquier otro caso:
Indicador de arrastre a 0
A = código del error

Siempre:
BC, DE, HL, IX corruptos
Preservados todos los demás registros

DD FORMATEAR
016Ch (364)

• Formatear una pista. (Utiliza la orden de formatear pista del μPD765A.)

El tampón contiene, para cada sector, los siguientes cuatro bytes:

C — número de la pista	(0...39)
H — número de la cabeza	(siempre 0 en las unidades de una sola cara gestionadas por +3DOS)
R — número del sector	(0...255)
N — $\log_2$(tamaño de sector)−7	(2 para sectores de 512 bytes)

Condiciones de entrada

B = página que entra en C000h...FFFFh (49152...65535)
C = unidad (0 o 1)
D = pista lógica (base 0)
E = byte para rellenar, generalmente E5h (229)
HL = dirección del tampón de formato
IX = dirección del XDPB

Situación de salida

Éxito:
Indicador de arrastre a 1
A corrupto

Fracaso:
Indicador de arrastre a 0
A = código del error

Siempre:
BC, DE, HL, IX corruptos
Preservados todos los demás registros

DD LEER ID
016Fh (367)

• Leer el identificador de un sector.

Condiciones de entrada

C = unidad (0 o 1)
D = pista lógica (base 0)
IX = dirección del XDPB

Situación de salida

Éxito:
Indicador de arrastre a 1
A = número del sector

Fracaso:
Indicador de arrastre a 0
A = código del error

Siempre:
HL = dirección del tampón resultante (en la página 7)
BC, DE, IX corruptos
Preservados todos los demás registros

DD PROBAR DISCO
0172h (370)

• Averiguar si el disco es de tipo adecuado.

Un disco de pista sencilla no es válido para una unidad de pista doble, y viceversa.

Condiciones de entrada

C = unidad (0 o 1)
IX = dirección del XDPB

Situación de salida

Éxito:
Indicador de arrastre a 1
A corrupto

Fracaso:
Indicador de arrastre a 0
A = código del error

Siempre:
BC, DE, HL, IX corruptos
Preservados todos los demás registros

DD RECONOCER
0175h (373)

- Reconocer el disco de la unidad.
- Inicializar el XDPB.

Esta rutina no afecta al indicador de congelación, ni lo tiene en cuenta.

Condiciones de entrada

C = unidad (0 o 1)
IX = dirección del XDPB de destino

Situación de salida

Éxito:
Indicador de arrastre a 1
A = tipo de disco
DE = tamaño del vector de asignación
HL = tamaño de la tabla de comprobación

Fracaso:
Indicador de arrastre a 0
A = código del error
DE, HL corruptos

Siempre:
BC, IX corruptos
Preservados todos los demás registros

DD SELEC FORMATO
0178h (376)

- Inicializar el XDPB para un formato estándar.

Esta rutina no afecta al indicador de congelación, ni lo tiene en cuenta.

Condiciones de entrada

A = tipo de disco
- 0 = Spectrum **+3** (Serie AMSTRAD PCW, DD UC PS)
- 1 = AMSTRAD CPC, formato de sistema
- 2 = AMSTRAD CPC, formato de datos
- 3 = AMSTRAD PCW, DD DC PD
- (otros valores = error)

IX = dirección del XDPB

Situación de salida

Éxito:
- Indicador de arrastre a 1
- A = tipo de disco
- DE = tamaño del vector de asignación de 2 bits
- HL = tamaño de la tabla de comprobación

Fracaso:
- Indicador de arrastre a 0
- A = código del error
- DE, HL corruptos

Siempre:
- BC, IX corruptos
- Preservados todos los demás registros

DD BUSCAR 1
017Bh (379)

- Averiguar si está presente la unidad 1. (Esta información se encuentra también en el bit 5 de la variable de sistema FLAGS3 de BASIC.)
- Poner en marcha el motor.
- Leer el estado de la unidad.

Si la unidad está 'no preparada' y 'protegida contra escritura', se considera que está ausente y entonces se inicia el descuento del 'tiempo de desconexión' del motor.

Esta rutina puede confundirse si el disco no está completamente introducido en la unidad.

Esta rutina supone que si no hay ningún disco en la unidad la 'protección contra escritura' tiene el valor 'verdadero', lo cual es cierto para las unidades de 3 y 8 pulgadas, pero no para las de 5¼.

Condiciones de entrada

Ninguna

Situación de salida

Si está presente la unidad 1:
Indicador de arrastre a 1

Si no lo está:
Indicador de arrastre a 0

Siempre:
A, BC, DE, HL, IX corruptos
Preservados todos los demás registros

DD ESTADO UNIDAD
017Eh (382)

- Leer el estado de la unidad.

Condiciones de entrada

C = unidad/cabeza
bits 0...1 = unidad
bit 2 = cabeza
bits 3..7 = 0

Situación de salida

A = ST3 (registro de estado 3 del μPD765A)

Siempre:
F, BC, DE, HL, IX corruptos
Preservados todos los demás registros

DD EQUIPO
0181h (385)

- Averiguar de qué tipo es la unidad (pista sencilla o doble, una o dos caras).

La información sobre las pistas sólo puede ser obtenida cuando el sistema ha «visto» el disco y ha identificado su tipo en la operación de reconocimiento.

La información sobre las caras sólo puede ser obtenida cuando el sistema ha «visto» un disco de dos caras y ha identificado su tipo en la operación de reconocimiento.

Condiciones de entrada

C = unidad (0 o 1)
IX = dirección del XDPB

Situación de salida

A = información sobre caras/pistas
bits 0...1 = información sobre caras
0 = desconocido
1 = una cara
2 = dos caras
bits 2...3 = información sobre pistas
0 = desconocido
1 = pista sencilla
2 = pista doble

Siempre:
F, BC, DE, HL, IX corruptos
Preservados todos los demás registros

DD PROT
0184h (388)

- Establecer la subrutina CODIF de protección contra copias.

Los discos protegidos contra copias tienen algunos de sus números de pista y de sector codificados en el propio disco. Antes de acceder al disco se ejecuta la subrutina CODIF para codificar los números de pista y de sector físicos. Los números así codificados tienen que coincidir con los que se encuentran en el sector de identificación.

No se debe codificar las pistas 0...2 de ninguna de las dos caras.

Condiciones de entrada

A = habilitar/inhibir
00h (0) = inhibir
FFh (255) = habilitar
HL = dirección de la subrutina CODIF (en caso de habilitar)

Situación de salida

HL = dirección de la anterior subrutina CODIF (0 si no existía)

Siempre:
AF, BC, DE, IX corruptos
Preservados todos los demás registros

Nota

La definición de la rutina CODIF es la que damos a continuación. En caso de existir una subrutina CODIF de usuario, las 'condiciones de entrada' son las condiciones en que se debe entrar en ella, mientras que la 'situación de salida' indica qué valores debe generar y qué registros le está permitido corromper.

CODIF

Condiciones de entrada

C = unidad/cara
bits 0...1 = unidad
bit 2 = cara
bits 3...7 = 0
D = pista física
E = sector físico
IX = dirección del DPB

Situación de salida

D = pista física codificada
E = sector físico codificado

Siempre:
AF corrupto
Preservados todos los demás registros

DD L XDPB
0187h (391)

- Inicializar un XDPB para un formato dado.

Esta rutina no afecta al indicador de congelación, ni lo tiene en cuenta.

Condiciones de entrada

IX = dirección del XDPB de destino
HL = dirección de la especificación de disco

Situación de salida

Éxito:
Indicador de arrastre a 1
A = tipo de disco grabado en el disco
DE = tamaño del vector de asignación
HL = tamaño de la tabla de comprobación

Fracaso:
  Indicador de arrastre a 0
  A = código del error
  DE, HL corruptos
Siempre:
  BC, IX corruptos
  Preservados todos los demás registros

DD L DPB
018Ah (394)

- Inicializar un DPB para un formato dado.

Esta rutina no afecta al indicador de congelación, ni lo tiene en cuenta.

Condiciones de entrada

IX = dirección del DPB de destino
HL = dirección de la especificación de disco

Situación de salida

Éxito:
  Indicador de arrastre a 1
  A = tipo de disco grabado en el disco
  DE = tamaño del vector de asignación
  HL = tamaño de la tabla de comprobación
Fracaso:
  Indicador de arrastre a 0
  A = código del error
  DE, HL corruptos
Siempre:
  BC, IX corruptos
  Preservados todos los demás registros

DD L BUSCAR
018Dh (397)

- Buscar hasta la pista requerida.
- Reintentar si es necesario.

Condiciones de entrada

C = unidad/cara
    bits 0...1 = unidad
    bit 2 = cara
    bits 3...7 = 0
D = pista
IX = dirección del XDPB

Situación de salida

Éxito:
  Indicador de arrastre a 1
  A corrupto
Fracaso:
  Indicador de arrastre a 0
  A = código del error
Siempre:
  BC, DE, HL, IX corruptos
  Preservados todos los demás registros

DD L LEER
0190h (400)

- Orden de lectura de bajo nivel del μPD765A.
- Leer datos.
- Leer datos borrados.
- Leer una pista.

Formato del bloque de parámetros:

Byte 0 — Página que entra en C000h...FFFFh (49152...65535)
Bytes 1...2 — Dirección del tampón
Bytes 3...4 — Número de bytes que deben ser transferidos
Byte 5 — Número de bytes de órdenes
Bytes 6... — Bytes de órdenes

Esta rutina escribe órdenes, lee datos, lee resultados.

El motor tiene que estar en marcha.

Condiciones de entrada

HL = dirección del bloque de parámetros

Situación de salida

HL = dirección del tampón resultante (en la página 7)

Siempre:
AF, BC, DE, IX corruptos
Preservados todos los demás registros

DD L ESCRIBIR
0193h (403)

- Orden de escritura de bajo nivel del μPD765A.
- Escribir datos.
- Escribir datos borrados.
- Formatear una pista.
- Verificar 'igual'.
- Verificar 'bajo o igual'.
- Verificar 'alto o igual'.

Formato del bloque de parámetros:

Byte 0 — Página que entra en C000h...FFFFh (49152...65535)
Bytes 1...2 — Dirección del tampón
Bytes 3...4 — Número de bytes que deben ser transferidos
Byte 5 — Número de bytes de órdenes
Bytes 6... — Bytes de órdenes

Esta rutina escribe órdenes, escribe datos, lee resultados.

El motor tiene que estar en marcha.

Condiciones de entrada

HL = dirección del bloque de parámetros

Situación de salida

HL = dirección del tampón resultante (en la página 7)

Siempre:
AF, BC, DE, IX corruptos
Preservados todos los demás registros

DD L MOTOR MARCHA
0196h (406)

- Poner en marcha el motor.
- Dejar que transcurra el 'tiempo de espera después de la puesta en marcha' del motor (establecido por DD CONFIGURAR).

Condiciones de entrada

Ninguna

Situación de salida

Siempre:
AF, BC, DE, HL, IX corruptos
Preservados todos los demás registros

DD L MOTOR TEMP
0199h (409)

- Iniciar el descuento del 'tiempo de espera después de la parada' del motor.

Condiciones de entrada

Ninguna

Situación de salida

Siempre:
AF, BC, DE, HL, IX corruptos
Preservados todos los demás registros

DD L MOTOR DESC
019Ch (412)

- Desconectar el motor.

Condiciones de entrada

Ninguna

Situación de salida

Siempre:
AF, BC, DE, HL, IX corruptos
Preservados todos los demás registros

Sección 28
Juego de caracteres del Spectrum

Temas tratados:

Códigos de control
Caracteres
Nemónicos de ensamblador en el Z80

La tabla siguiente da la lista completa del juego de caracteres del Spectrum, con sus códigos en versión decimal y hexadecimal. Por otra parte, si se considera esos códigos como instrucciones de código de máquina del Z80, entonces las columnas de la derecha dan los nemónicos correspondientes en lenguaje ensamblador. Tenga en cuenta que algunas instrucciones del Z80 empiezan por CBh o EDh; éstas aparecen en las dos columnas de la derecha.

Cuando un carácter es distinto en las dos versiones de BASIC (48k y 128k), el carácter correspondiente a 48 BASIC se escribe entre paréntesis a continuación del de +BASIC.

Código	Carácter	Hex	Ensamblador	Tras CBh	Tras EDh
0	no utilizado	00	**nop**	**cb**	
1	no utilizado	01	**ld bc,NN**	**rlc c**	
2	no utilizado	02	**ld(bc),a**	**rlc d**	
3	no utilizado	03	**inc bc**	**rlc e**	
4	no utilizado	04	**inc b**	**rlc h**	
5	no utilizado	05	**dec b**	**rlc l**	
6	coma de **PRINT**	06	**ld b,N**	**rlc (hl)**	
7	EDIT	07	**rlca**	**rlc a**	
8	cursor a la izquierda, ←	08	**ex af,af**	**rrc b**	
9	cursor a la derecha, →	09	**add hl,bc**	**rrc c**	
10	cursor abajo, ↓	0A	**ld a,(bc)**	**rrc d**	
11	cursor arriba, ↑	0B	**dec bc**	**rrc e**	
12	BORRAR	0C	**inc c**	**rrc h**	
13	INTRO	0D	**dec c**	**rrc l**	
14	número	0E	**ld c,N**	**rrc (hl)**	
15	no utilizado	0F	**rrca**	**rrc a**	
16	control de **INK**	10	**djnz DIS**	**rl b**	
17	control de **PAPER**	11	**ld de,NN**	**rl c**	

Código	Carácter	Hex	Ensamblador	Tras CBh	Tras EDh
18	control de **FLASH**	12	**ld (de),a**	**rl d**	
19	control de **BRIGHT**	13	**inc de**	**rl e**	
20	control de **INVERSE**	14	**inc d**	**rl h**	
21	control de **OVER**	15	**dec d**	**rl l**	
22	control de **AT**	16	**ld d,N**	**rl (hl)**	
23	control de **TAB**	17	**rla**	**rl a**	
24	no utilizado	18	**jr DIS**	**rr b**	
25	no utilizado	19	**add hl,de**	**rr c**	
26	no utilizado	1A	**ld a,(de)**	**rr d**	
27	no utilizado	1B	**dec de**	**rr e**	
28	no utilizado	1C	**inc e**	**rr h**	
29	no utilizado	1D	**dec e**	**rr l**	
30	no utilizado	1E	**ld e,N**	**rr (hl)**	
31	no utilizado	1F	**rra**	**rr a**	
32	espacio	20	**jr nz,DIS**	**sla b**	
33	!	21	**ld hl,NN**	**sla c**	
34	"	22	**ld(NN),hl**	**sla d**	
35	#	23	**inc hl**	**sla e**	
36	$	24	**inc h**	**sla h**	
37	%	25	**dec h**	**sla l**	
38	&	26	**ld h,N**	**sla (hl)**	
39	'	27	**daa**	**sla a**	
40	(	28	**jr z,DIS**	**sra b**	
41	)	29	**add hl,hl**	**sra c**	
42	*	2A	**ld hl,(NN)**	**sra d**	
43	+	2B	**dec hl**	**sra e**	
44	,	2C	**inc l**	**sra h**	
45	−	2D	**dec l**	**sra l**	
46	.	2E	**ld l,N**	**sra (hl)**	
47	/	2F	**cpl**	**sra a**	
48	0	30	**jr nc,DIS**		
49	1	31	**ld sp,NN**		
50	2	32	**ld (NN),a**		
51	3	33	**inc sp**		
52	4	34	**inc (hl)**		
53	5	35	**dec (hl)**		
54	6	36	**ld (hl),N**		
55	7	37	**scf**		
56	8	38	**jr c,DIS**	**srl b**	
57	9	39	**add hl,sp**	**srl c**	
58	:	3A	**ld a,(NN)**	**srl d**	

Código	Carácter	Hex	Ensamblador	Tras CBh	Tras EDh
59	;	3B	dec sp	srl e	
60	<	3C	inc a	srl h	
61	=	3D	dec a	srl l	
62	>	3E	ld a,N	srl (hl)	
63	?	3F	ccf	srl a	
64	@	40	ld b,b	bit 0,b	in b,(c)
65	A	41	ld b,c	bit 0,c	out (c),b
66	B	42	ld b,d	bit 0,d	sbc hl,bc
67	C	43	ld b,e	bit 0,e	ld (NN),bc
68	D	44	ld b,h	bit 0,h	neg
69	E	45	ld b,l	bit 0,l	retn
70	F	46	ld b,(hl)	bit 0,(hl)	im 0
71	G	47	ld b,a	bit 0,a	ld i,a
72	H	48	ld c,b	bit 1,b	in c,(c)
73	I	49	ld c,c	bit 1,c	out (c),c
74	J	4A	ld c,d	bit 1,d	adc hl,bc
75	K	4B	ld c,e	bit 1,e	ld bc,(NN)
76	L	4C	ld c,h	bit 1,h	
77	M	4D	ld c,l	bit 1,l	reti
78	N	4E	ld c,(hl)	bit 1,(hl)	
79	O	4F	ld c,a	bit 1,a	ld r,a
80	P	50	ld d,b	bit 2,b	in d,(c)
81	Q	51	ld d,c	bit 2,c	out (c),d
82	R	52	ld d,d	bit 2,d	sbc hl,de
83	S	53	ld d,e	bit 2,e	ld (NN),de
84	T	54	ld d,h	bit 2,h	
85	U	55	ld d,l	bit 2,l	
86	V	56	ld d,(hl)	bit 2,(hl)	im 1
87	W	57	ld d,a	bit 2,a	ld a,i
88	X	58	ld e,b	bit 3,b	in e,(c)
89	Y	59	ld e,c	bit 3,c	out (c),e
90	Z	5A	ld e,d	bit 3,d	adc hl,de
91	¡	5B	ld e,e	bit 3,e	ld de,(NN)
92	Ñ	5C	ld e,h	bit 3,h	
93	¿	5D	ld e,l	bit 3,l	
94	↑	5E	ld e,(hl)	bit 3,(hl)	im 2
95	_	5F	ld e,a	bit 3,a	ld a,r
96	Pt	60	ld h,b	bit 4,b	in h,(c)
97	a	61	ld h,c	bit 4,c	out (c),h
98	b	62	ld h,d	bit 4,d	sbc hl,hl
99	c	63	ld h,e	bit 4,e	ld (NN),hl

Código	Carácter	Hex	Ensamblador	Tras CBh	Tras EDh
100	d	64	**ld h,h**	**bit 4,h**	
101	e	65	**ld h,l**	**bit 4,l**	
102	f	66	**ld h,(hl)**	**bit 4,(hl)**	
103	g	67	**ld h,a**	**bit 4,a**	**rrd**
104	h	68	**ld l,b**	**bit 5,b**	**in l,(c)**
105	i	69	**ld l,c**	**bit 5,c**	**out (C),l**
106	j	6A	**ld l,d**	**bit 5,d**	**adc hl,hl**
107	k	6B	**ld l,e**	**bit 5,e**	**ld hl,(NN)**
108	l	6C	**ld l,h**	**bit 5,h**	
109	m	6D	**ld l,l**	**bit 5,l**	
110	n	6E	**ld l,(hl)**	**bit 5,(hl)**	
111	o	6F	**ld l,a**	**bit 5,a**	**rld**
112	p	70	**ld (hl),b**	**bit 6,b**	**in f,(c)**
113	q	71	**ld (hl),c**	**bit 6,c**	
114	r	72	**ld (hl),d**	**bit 6,d**	**sbc hl,sp**
115	s	73	**ld (hl),e**	**bit 6,e**	**ld (NN),sp**
116	t	74	**ld (hl),h**	**bit 6,h**	
117	u	75	**ld (hl),l**	**bit 6,l**	
118	v	76	**halt**	**bit 6,(hl)**	
119	w	77	**ld (hl),a**	**bit 6,a**	
120	x	78	**ld a,b**	**bit 7,b**	**in a,(c)**
121	y	79	**ld a,c**	**bit 7,c**	**out (c),a**
122	z	7A	**ld a,d**	**bit 7,d**	**adc hl,sp**
123	{	7B	**ld a,e**	**bit 7,e**	**ld sp,(NN)**
124	ñ	7C	**ld a,h**	**bit 7,h**	
125	}	7D	**ld a,l**	**bit 7,l**	
126	~	7E	**ld a,(hl)**	**bit 7,(hl)**	
127	©	7F	**ld a,a**	**bit 7,a**	
128		80	**add a,b**	**res 0,b**	
129		81	**add a,c**	**res 0,c**	
130		82	**add a,d**	**res 0,d**	
131		83	**add a,e**	**res 0,e**	
132		84	**add a,h**	**res 0,h**	
133		85	**add a,l**	**res 0,l**	
134		86	**add a,(hl)**	**res 0,(hl)**	
135		87	**add a,a**	**res 0,a**	
136		88	**adc a,b**	**res 1,b**	
137		89	**adc a,c**	**res 1,c**	
138		8A	**adc a,d**	**res 1,d**	
139		8B	**adc a,e**	**res 1,e**	
140		8C	**adc a,h**	**res 1,h**	

Código	Carácter		Hex	Ensamblador	Tras CBh	Tras EDh
141			8D	**adc a,l**	**res 1,l**	
142			8E	**adc a,(hl)**	**res 1,(hl)**	
143			8F	**adc a,a**	**res 1,a**	
144	(a)		90	**sub b**	**res 2,b**	
145	(b)		91	**sub c**	**res 2,c**	
146	(c)		92	**sub d**	**res 2,d**	
147	(d)		93	**sub e**	**res 2,e**	
148	(e)		94	**sub h**	**res 2,h**	
149	(f)		95	**sub l**	**res 2,l**	
150	(g)		96	**sub (hl)**	**res 2,(hl)**	
151	(h)		97	**sub a**	**res 2,a**	
152	(i)		98	**sbc a,b**	**res 3,b**	
153	(j)	gráficos de	99	**sbc a,c**	**res 3,c**	
154	(k)	usuario	9A	**sbc a,d**	**res 3,d**	
155	(l)		9B	**sbc a,e**	**res 3,e**	
156	(m)		9C	**sbc a,h**	**res 3,h**	
157	(n)		9D	**sbc a,l**	**res 3,l**	
158	(o)		9E	**sbc a,(hl)**	**res 3,(hl)**	
159	(p)		9F	**sbc a,a**	**res 3,a**	
160	(q)		A0	**and b**	**res 4,b**	**ldi**
161	(r)		A1	**and c**	**res 4,c**	**cpi**
162	(s)		A2	**and d**	**res 4,d**	**ini**
163	SPECTRUM (t)		A3	**and e**	**res 4,e**	**outi**
164	PLAY (u)		A4	**and h**	**res 4,h**	
165	RND		A5	**and l**	**res 4,l**	
166	INKEY$		A6	**and (hl)**	**res 4,(hl)**	
167	PI		A7	**and a**	**res 4,a**	
168	FN		A8	**xor b**	**res 5,b**	**ldd**
169	POINT		A9	**xor c**	**res 5,c**	**cpd**
170	SCREEN$		AA	**xor d**	**res 5,d**	**ind**
171	ATTR		AB	**xor e**	**res 5,e**	**outd**
172	AT		AC	**xor h**	**res 5,h**	
173	TAB		AD	**xor l**	**res 5,l**	
174	VAL$		AE	**xor (hl)**	**res 5,(hl)**	
175	CODE		AF	**xor a**	**res 5,a**	
176	VAL		B0	**or b**	**res 6,b**	**ldir**
177	LEN		B1	**or c**	**res 6,c**	**cpir**
178	SIN		B2	**or d**	**res 6,d**	**inir**
179	COS		B3	**or e**	**res 6,e**	**otir**
180	TAN		B4	**or h**	**res 6,h**	
181	ASN		B5	**or l**	**res 6,l**	

Código	Carácter	Hex	Ensamblador	Tras CBh	Tras EDh
182	ACS	B6	or (hl)	res 6,(hl)	
183	ATN	B7	or a	res 6,a	
184	LN	B8	cp b	res 7,b	lddr
185	EXP	B9	cp c	res 7,c	cpdr
186	INT	BA	cp d	res 7,d	indr
187	SQR	BB	cp e	res 7,e	otdr
188	SGN	BC	cp h	res 7,h	
189	ABS	BD	cp l	res 7,l	
190	PEEK	BE	cp (hl)	res 7,(hl)	
191	IN	BF	cp a	res 7,a	
192	USR	C0	ret nz	set 0,b	
193	STR$	C1	pop bc	set 0,c	
194	CHR$	C2	jp nz,NN	set 0,d	
195	NOT	C3	jp NN	set 0,e	
196	BIN	C4	call nz,NN	set 0,h	
197	OR	C5	push bc	set 0,l	
198	AND	C6	add a,N	set 0,(hl)	
199	<=	C7	rst 0	set 0,a	
200	>=	C8	ret z	set 1,b	
201	<>	C9	ret	set 1,c	
202	LINE	CA	jp z,NN	set 1,d	
203	THEN	CB		set 1,e	
204	TO	CC	call z,NN	set 1,h	
205	STEP	CD	call NN	set 1,l	
206	DEF FN	CE	adc a,N	set 1,(hl)	
207	CAT	CF	rst 8	set 1,a	
208	FORMAT	D0	ret nc	set 2,b	
209	MOVE	D1	pop de	set 2,c	
210	ERASE	D2	jp nc,NN	set 2,d	
211	OPEN #	D3	out (N),a	set 2,e	
212	CLOSE #	D4	call nc,NN	set 2,h	
213	MERGE	D5	push de	set 2,l	
214	VERIFY	D6	sub N	set 2,(hl)	
215	BEEP	D7	rst 16	set 2,a	
216	CIRCLE	D8	ret c	set 3,b	
217	INK	D9	exx	set 3,c	
218	PAPER	DA	jp c,NN	set 3,d	
219	FLASH	DB	in a,(N)	set 3,e	
220	BRIGHT	DC	call c,NN	set 3,h	

Código	Carácter	Hex	Ensamblador	Tras CBh	Tras EDh
221	INVERSE	DD	Prefijo de instrucciones que afectan a ix	set 3,l	
222	OVER	DE	sbc a,N	set 3,(hl)	
223	OUT	DF	rst 24	set 3,a	
224	LPRINT	E0	ret po	set 4,b	
225	LLIST	E1	pop hl	set 4,c	
226	STOP	E2	jp po,NN	set 4,d	
227	READ	E3	ex (sp),hl	set 4,e	
228	DATA	E4	call po,NN	set 4,h	
229	RESTORE	E5	push hl	set 4,l	
230	NEW	E6	and N	set 4,(hl)	
231	BORDER	E7	rst 32	set 4,a	
232	CONTINUE	E8	ret pe	set 5,b	
233	DIM	E9	jp (hl)	set 5,c	
234	REM	EA	jp pe,NN	set 5,d	
235	FOR	EB	ex de,hl	set 5,e	
236	GO TO	EC	call pe,NN	set 5,h	
237	GO SUB	ED		set 5,l	
238	INPUT	EE	xor N	set 5,(hl)	
239	LOAD	EF	rst 40	set 5,a	
240	LIST	F0	ret p	set 6,b	
241	LET	F1	pop af	set 6,c	
242	PAUSE	F2	jp p,NN	set 6,d	
243	NEXT	F3	di	set 6,e	
244	POKE	F4	call p,NN	set 6,h	
245	PRINT	F5	push af	set 6,l	
246	PLOT	F6	or N	set 6,(hl)	
247	RUN	F7	rst 48	set 6,a	
248	SAVE	F8	ret m	set 7,b	
249	RANDOMIZE	F9	ld sp,hl	set 7,c	
250	IF	FA	jp m,NN	set 7,d	
251	CLS	FB	ei	set 7,e	
252	DRAW	FC	call m,NN	set 7,h	
253	CLEAR	FD	Prefijo de instrucciones que afectan a iy	set 7,l	
254	RETURN	FE	cp N	set 7,(hl)	
255	COPY	FF	rst 56	set 7,a	

Sección 29
Mensajes

Temas tratados:

Informes y mensajes
CONTINUE

En la última línea de la pantalla aparece un mensaje cada vez que el **+2A** termina de ejecutar alguna orden en BASIC. Su misión es explicar a qué se debe la detención, ya sea por razones naturales o porque ha habido un error.

Casi todos los informes consisten en un código (número o letra), por el que se lo puede encontrar en la tabla siguiente, un breve texto que explica lo que ha ocurrido y el número de línea (y el número de sentencia dentro de esa línea) en la que se ha detenido. Cuando se trata de una orden directa, el número de línea que muestra es el 0. Dentro de una línea, la sentencia 1 es la que se encuentra al principio, la 2 es la que va a continuación del primer signo de dos puntos (o de **THEN**), etc.

Los mensajes relacionados con la unidad de disco no van precedidos de ningún código. En la tabla están en orden alfabético.

El comportamiento de la orden **CONTINUE** depende en gran medida de los mensajes. Normalmente **CONTINUE** va a la línea y sentencia indicadas en el último informe, pero hay excepciones en los mensajes de códigos **0**, **9** y **D**.

La siguiente tabla muestra todos los informes e indica en qué circunstancias pueden aparecer; con esta información se puede consultar la sección 31. Por ejemplo, el error **'K COLOR NO VALIDO'** puede presentarse en **INK, PAPER, BORDER, FLASH, BRIGHT, INVERSE** y **OVER**, y la descripción que se da de estas palabras clave en la Sección 31 indica exactamente qué números de color no son válidos para ellas.

Los mensajes marcados con **'RIC'** (en la primera columna) van normalmente seguidos de las opciones **'¿REINTENTAR, IGNORAR o CANCELAR'**. Si el usuario pulsa [C] para elegir **CANCELAR**, aparece el mensaje de la segunda columna.

Código	Texto y significado	Situación
0	**OK** Tarea terminada con éxito, o intento de saltar a un número de línea mayor que cualquiera de los existentes. Este mensaje no cambia la línea ni la sentencia a la que salta **CONTINUE**.	Cualquiera
1	**NEXT sin FOR** La variable de control no existe (no ha sido establecida por una sentencia **FOR**), pero hay una variable ordinaria con el mismo nombre.	**NEXT**
2	**VARIABLE NO DEFINIDA** En el caso de una variable sencilla, ocurre si se intenta utilizarla antes de definirla (con **LET**, **READ** o **INPUT**), o de cargarla desde la cinta o el disco, o de definirla en una sentencia **FOR**. En el caso de una variable indexada, ocurre si se intenta utilizarla antes de dimensionar la matriz (**DIM**) o de cargarla desde la cinta o el disco.	Cualquiera
3	**SUBINDICE INCORRECTO** Un subíndice es mayor que la dimensión de la matriz, o el número de subíndices no se corresponde con el de dimensiones de la matriz. Si el subíndice es negativo o mayor que 65535, se produce el error **B**.	Variables indexadas, subcadenas

Código	Texto y significado	Situación
4	**MEMORIA AGOTADA** No hay espacio suficiente en la memoria del ordenador para lo que se pretende hacer. Si el ordenador parece ser incapaz de salir de esta situación, será necesario borrar la línea de órdenes actual pulsando [BORRAR] y luego borrar una o dos líneas del programa (con la intención de reintroducirlas más tarde).	**LET, INPUT, FOR, DIM, GO SUB, LOAD, MERGE.** A veces durante la evaluación de una expresión
5	**FUERA DE PANTALLA** Una sentencia **INPUT** ha tratado de generar más de 23 líneas en la pantalla inferior. También se produce con **PRINT AT 22,*xx*.**	**INPUT, PRINT AT**
6	**NUMERO MUY GRANDE** Los cálculos han desembocado en un número superior a aproximadamente 10^{38}.	Cualquier cálculo aritmético
7	**RETURN sin GO SUB** Hay un **RETURN** al que no corresponde ningún **GO SUB.**	**RETURN**
9	**Sentencia STOP** Después de este mensaje, **CONTINUE** no repetirá la sentencia **STOP,** sino que continuará a partir de la sentencia siguiente.	**STOP**
A	**ARG. INVALIDO** El argumento que se ha puesto en una función no es adecuado.	**SQR, LN, ASN, ACS, USR** (con argumento literal)

Código	Texto y significado	Situación
B	**ENTERO EXCEDE MARGEN** Cuando una sentencia requiere un entero, el argumento de punto flotante es redondeado al entero más próximo. Este error se presenta si la operación de redondeo produce un entero fuera del margen correcto.	**RUN, RANDOMIZE, POKE, DIM, GO TO, GO SUB, LIST, LLIST, PAUSE, PLOT, CHR$, PEEK, USR** (con argumento numérico)
	En relación con las matrices, véase también el error **3**.	Acceso a matrices
C	**SIN SENTIDO EN BASIC** El texto del argumento (literal) no constituye una expresión válida. También se produce cuando el argumento de una función o de una orden es escandalosamente erróneo.	**VAL, VAL$**
D	**BREAK – CONT repite** Se ha pulsado BREAK durante una operación con algún periférico. El comportamiento de **CONTINUE** tras este informe es normal en el sentido de que repite la sentencia. Compárese esto con el informe **L**.	**LOAD, SAVE, VERIFY, MERGE.** También cuando el ordenador pregunta '**¿MAS?**' y se responde pulsando N, BREAK o la barra espaciadora
E	**DATOS AGOTADOS** Se ha tratado de leer más allá del final de la lista **DATA**.	**READ**
F	**NOMBRE INCORRECTO** El nombre especificado tras **SAVE** es la cadena vacía (o consta de más de 10 caracteres y **SAVE** está dirigida a la cinta).	**SAVE**

Código	Texto y significado	Situación
G	**NO CABE LA LINEA** No queda espacio en la memoria para acomodar la nueva línea de programa.	Introducción de una línea de programa
H	**STOP en INPUT** Algún dato introducido en **INPUT** comenzó con **STOP**. A diferencia del caso del informe **9**, después de **H** la orden **CONTINUE** se comporta normalmente, repitiendo la sentencia **INPUT**.	**INPUT**
I	**FOR sin NEXT** Se ha establecido un bucle que no tiene que ser ejecutado ninguna vez (por ejemplo, **FOR n=1 TO 0**) y BASIC no ha encontrado el **NEXT** correspondiente.	**FOR**
J	**DISP. E/S INCORRECTO** Se ha tratado de leer o escribir caracteres en un dispositivo de entrada/salida que no es capaz de gestionar la operación. Por ejemplo, no se puede captar caracteres por la pantalla. Una orden tal como **INPUT #2,a$** produciría este error.	Operaciones con canales: **OPEN #**, **CLOSE #**, **INPUT #**, **PRINT #**, etc.
K	**COLOR NO VALIDO** El número especificado no es correcto.	**INK, PAPER, BORDER, FLASH, BRIGHT, INVERSE, OVER**; también después de uno de los correspondientes caracteres de control

Código	Texto y significado	Situación
L	**PROGR. INTERRUMPIDO** Se ha pulsado [BREAK]. BASIC explora la posible pulsación de este tecla entre cada dos sentencias. Los números de línea y sentencia del informe se refieren a la sentencia anterior a la pulsación de [BREAK], pero **CONTINUE** va a la sentencia siguiente, de forma que no repite ninguna.	Cualquiera
M	**RAMTOP INCORRECTA** El número especificado para RAMTOP es demasiado grande o demasiado pequeño.	**CLEAR**; posiblemente también en **RUN**
N	**SENT. PERDIDA** Salto a una sentencia que ya no existe.	**RETURN, NEXT, CONTINUE**
O	**CANAL NO VALIDO** Se ha tratado de acceder a un canal que no está abierto o cuyo número está fuera de margen (0. . .15), o se ha tratado de abrir un canal con un número fuera de margen.	**OPEN #, INPUT #, PRINT #**
P	**FN sin DEF** Se ha intentado utilizar una función de usuario que no está definida en el programa.	**FN**
Q	**PARAMETRO ERRONEO** Número de argumentos incorrecto, o bien uno de ellos es de tipo incorrecto (cadena en vez de número, o viceversa).	**FN**

Código	Texto y significado	Situación
R	**ERROR CARGANDO CINTA** Se ha encontrado el fichero en la cinta, pero por alguna razón no ha sido posible leerlo o verificarlo.	**VERIFY, LOAD, MERGE**
d	**EXCESO DE PARENTESIS** Demasiados paréntesis en una frase repetida en uno de los argumentos.	**PLAY**
j	**VELOCIDAD NO VALIDA** Se ha especificado velocidad cero para la puerta RS232.	**FORMAT LINE**
k	**NOTA NO VALIDA** **PLAY** ha encontrado una nota o una orden que no reconoce, o una orden escrita en minúsculas.	**PLAY**
l	**NUMERO MUY GRANDE** El parámetro de una orden de **PLAY** es un orden de magnitud demasiado grande.	**PLAY**
m	**NOTA FUERA DE MARGEN** Una serie de bemoles o sostenidos ha producido una nota que excede del margen permitido por el circuito de sonido.	**PLAY**
n	**FUERA DE MARGEN** El parámetro de una orden es demasiado grande o demasiado pequeño. Si es muy grande, se produce el error **l**.	**PLAY**

Código	Texto y significado	Situación
o	**DEMASIADAS NOTAS** Se ha intentado hacer una ligadura con demasiadas notas.	**PLAY**
	ATRIBUTO NO VALIDO La especificación de atributo que se ha escrito tras + o – en una orden **MOVE** no es **P**, **S** ni **A**.	**MOVE...TO**
	DESTINO NO ES UNIDAD En una orden **COPY** se ha usado caracteres polivalentes en la especificación de fichero de origen, pero la de destino es un nombre de fichero concreto. Cuando la especificación de origen es una plantilla, la de destino solamente puede consistir en la letra de la unidad y no puede contener caracteres polivalentes.	**COPY...TO**
	DESTINO POLIVALENTE Se ha incluido caracteres polivalentes en las dos especificaciones de fichero de una orden **COPY**. Cuando la especificación de origen es una plantilla, la de destino solamente puede consistir en la letra de la unidad y no puede contener caracteres polivalentes.	**COPY...TO**
	DIRECTORIO LLENO Se ha tratado de crear un nuevo fichero cuando el directorio ya contenía todas las reseñas posibles (en un disco normal, 64).	**COPY, SAVE**

Código	Texto y significado	Situación
RIC	**DISCO CAMBIADO** Al ejecutar una de estas órdenes, +3DOS ha descubierto que el disco que se encuentra en la unidad no es el mismo que cuando se inició la ejecución de la orden. Este error se produce si una orden de +3 BASIC trata de acceder a un disco en el que un programa de código de máquina ha abierto algún fichero.	**CAT, COPY, ERASE, LOAD, MERGE, MOVE, SAVE**
	DISCO LLENO Se ha tratado de crear un nuevo fichero en un disco en el que no queda espacio libre. Antes de realizar esta operación se puede dar una orden **CAT** para averiguar cuánto espacio queda. Si el error se produce en una orden **COPY**, ésta borrará el fichero que ha copiado parcialmente. Sin embargo, **SAVE** puede dejar en el disco un fichero incompleto; el usuario debe borrarlo, pues todo intento de utilizarlo fracasaría.	**COPY, SAVE**
	DISCO NO DE ARRANQUE Se ha intentado cargar un sector de inicialización con un disco que no dispone de ese sector.	**LOAD "*"**
RIC	**DISCO PROTEGIDO** Se ha intentado escribir en un disco cuyo orificio de protección contra escritura está abierto. Para desactivar la protección se debe obturar el orificio.	**COPY, ERASE, FORMAT, MOVE, SAVE**

Código	Texto y significado	Situación
RIC	**ERROR DE DATOS** La comprobación de redundancia cíclica (CRC, suma de comprobación) para un sector es incorrecta. Este error puede presentarse si el disco ha sido alterado (quizá magnéticamente).	**CAT, COPY, ERASE, LOAD, MERGE, MOVE, SAVE**
	ERROR DE LONGITUD Se ha tratado de cargar un fichero de tipo '**CODE**' cuya longitud es mayor que la especificada en **LOAD**.	**LOAD. . .CODE**
RIC	**ERROR DESCONOCIDO** Se ha producido un error que el sistema no tiene previsto. Es muy improbable que el usuario llegue a ver este mensaje.	Improbable; **CAT, COPY, ERASE, FORMAT, LOAD, MERGE, MOVE, SAVE**
RIC	**FALLO DE BUSQUEDA** Es un error de hardware que significa que la unidad no ha sido capaz de localizar la pista que se le había solicitado. Si persiste este error, es posible que haya que reparar la máquina.	**CAT, COPY, ERASE, FORMAT, LOAD, MERGE, MOVE, SAVE**
	FALTA CACHE Se trata de un error interno del sistema. Es muy improbable que el usuario llegue a ver este mensaje.	Improbable

Código	Texto y significado	Situación
	FALTA SECCION Los ficheros de disco consisten en bloques de 16K, cada uno de los cuales se denomina *sección*. Este error puede ocurrir si se cambia el disco después de que el sistema haya leído la reseña de un fichero en el directorio y antes de que haya leído su primera sección. Sin embargo, es muy improbable que el usuario llegue a ver este error.	Improbable; **COPY, LOAD, MERGE**
	FICH. DEMASIADO GRANDE Se ha intentado escribir en un fichero cuya longitud es mayor que 8 megabytes. Es muy improbable que se produzca este error.	Improbable
	FICHERO NO ABIERTO Una orden de disco ha intentado acceder a un fichero que no ha sido abierto previamente. Es muy improbable que se produzca este error.	Improbable
	FICHERO NO ENCONTRADO Se ha especificado un nombre de fichero que no existe en el disco.	**COPY,ERASE,LOAD,MERGE,MOVE**
	FICHERO SOLO LECTURA Se ha intentado ejecutar una de estas órdenes usando el nombre de un fichero que tiene activado el atributo de protección (con **MOVE** *nombre-de-fichero* **TO "+P"**). Para desprotegerlo se debe ejecutar la orden **MOVE** *nombre-de-fichero* **TO "−P"**.	**COPY, ERASE, MOVE, SAVE**

Código	Texto y significado	Situación
	FICHERO YA EN USO Este error se produce si un programa de código de máquina ha abierto un fichero con número 1...3 y una orden de +3 BASIC trata de abrirlo otra vez. Es un error muy improbable.	Improbable; **COPY, LOAD, MERGE, SAVE**
	FIN DE FICHERO Se ha intentado leer un byte más allá de la posición de fin de fichero. No es probable que se produzca este error.	Improbable
RIC	**FORMATO NO RECONOCIDO** Al tratar de leer o escribir en un disco, +3DOS no ha podido reconocer su formato. Es decir, ha leído la especificación del disco, pero no ha entendido esa información. Este error puede ocurrir al usar discos dotados de alguna protección especial.	**CAT, COPY, ERASE, LOAD, MERGE, MOVE, SAVE**
	NOMBRE INCORRECTO El nombre de fichero especificado en una orden de disco no se ajusta a las normas descritas en la Sección 20 de este capítulo.	**CAT, COPY, ERASE, FORMAT, LOAD, MERGE, MOVE, SAVE**
	PARAMETRO INCORRECTO Un valor entregado por BASIC a +3DOS está fuera de margen. No es probable que se produzca este error.	Improbable

Código	Texto y significado	Situación
RIC	**SIN DATOS** Es un error de disco de bajo nivel que se produce cuando el sistema no encuentra el identificador de un sector. Puede ocurrir si se intenta copiar un disco que tenga alguna protección especial.	**CAT, COPY, ERASE, LOAD, MERGE, MOVE, SAVE**
RIC	**SIN MARCA DIRECCIONES** El sector que se estaba leyendo no contiene la información que identifica su posición en el disco. Casi siempre se produce al intentar leer un disco no formateado. También puede ser causado por una corrupción de la información del disco o por algún sistema de protección especial.	**CAT, COPY, ERASE, FORMAT, LOAD, MERGE, MOVE, SAVE**
	SINTAXIS INCORRECTA Las especificaciones de origen y destino en una orden **MOVE** hacen referencia a unidades diferentes. No se puede usar la orden **MOVE** para trasladar un fichero de una unidad a otra.	**MOVE...TO**
RIC	**SOPORTE NO ADECUADO** El disco tiene un formato no adecuado a la unidad. Este error se presenta, por ejemplo, cuando se intenta escribir en un disco de 80 pistas con la unidad AMSTRAD FD-1, que es de 40 pistas.	**CAT, COPY, ERASE, FORMAT, LOAD, MERGE, MOVE, SAVE**

Código	Texto y significado	Situación
	UNIDAD B AUSENTE Se ha intentado aplicar la orden **FORMAT** a la segunda unidad de disco externa (unidad B), pero no está conectada.	**FORMAT**
	UNIDAD INCORRECTA La letra de unidad especificada en una orden **FORMAT** no es **A** ni **B**, o bien se ha intentado seleccionar como unidad implícita una unidad que no existe.	**FORMAT**
	UNIDAD NO ENCONTRADA La especificación de fichero incluye la letra de una unidad que no está presente en el sistema. Por ejemplo, **ERASE "c:banco"**.	**CAT, COPY, ERASE, LOAD, MERGE, MOVE, SAVE**
RIC	**UNIDAD NO PREPARADA** Se ha intentado ejecutar una orden de disco cuando la unidad no estaba preparada. Esto ocurre generalmente cuando no se ha introducido ningún disco en la unidad. Normalmente se podrá introducir el disco y pulsar [R].	**CAT, COPY, ERASE, FORMAT, LOAD, MERGE, MOVE, SAVE**
	UNIDAD YA EN USO Se ha intentado reproyectar una unidad en la que hay ficheros abiertos. No es probable que se produzca este error.	Improbable

Código	Texto y significado	Situación
	YA EXISTE EL FICHERO Ya existe en el disco un fichero que tiene el nombre especificado en una orden **MOVE** como nombre de destino.	**MOVE...TO**
	+2A NO ADMITE FORMAT Este error se produce cuando se intenta ejecutar una orden **FORMAT** sin tener conectada la unidad de disquete externa.	**FORMAT**

Sección 30
Información de referencia

Temas tratados:

Hardware

El **+2A** está diseñado en torno al microprocesador Z80, el cual funciona a una velocidad de 3.5469 MHz (unos tres millones y medio de ciclos por segundo).

La memoria del **+2A** está dividida en 64K de ROM y 128K de RAM y distribuida en páginas de 16K. A las cuatro páginas de ROM (0-3) se accede a través de los 16K inferiores (direcciones 0000h a 3FFFh) del mapa de la memoria. Las ocho páginas de RAM (0-7) son accesibles a través de los 16K superiores del mapa (direcciones C000h a FFFFh). La página 5 de la RAM también puede ser encajada en el margen de 4000h a 7FFFh, y la página 2 en el comprendido entre 8000h y BFFFh. Además, hay varias formas de ocupar los 64K con páginas de RAM; las combinaciones posibles han sido descritas en la Sección 24, apartado 'Gestión de la memoria'.

Físicamente hablando, las ROM se encuentran en dos dispositivos de 32K cada uno (similares al 27256) que el sistema trata como si fuesen dos circuitos de 16K. La RAM está compuesta por dieciséis circuitos de 64K × 4 bits (41464), algunos de los cuales (bancos RAM4 a RAM7) están compartidos entre los circuitos que producen la imagen de pantalla y el Z80A. Los otros circuitos (bancos RAM0 a RAM3), así como la ROM, son de uso exclusivo del Z80A.

El sistema de compartición de RAM (entre los circuitos de video y el procesador) funciona del siguiente modo:

- Cada cuadro de TV consta de 311 líneas.
- De ellas, sólo hay conflicto en 192.
- Cada línea de TV dura 228 estados T.
- De éstos, sólo hay conflicto en 128 estados T.
- Durante estos 128 estados T, al procesador sólo se le permite usar 1 de cada 8; los otros 7 están dedicados al controlador de video.

El efecto global es una reducción de la frecuencia de 4 a 2.66 MHz.

La matriz lógica no comprometida (ULA, *Uncommitted Logic Array*) controla la mayor parte de las operaciones de E/S: teclado, interfaz del magnetófono, pantalla y parte del interfaz de la impresora. Convierte los bytes que se encuentran en la memoria en formas y colores para la pantalla y permite al Z80A explorar el teclado y leer y escribir los datos en la cinta.

El sonido en tres canales es producido por el AY-3-8912, un circuito de sonido muy popular; este dispositivo controla también las puertas **RS232/MIDI** y **AUX**.

Las dos puertas serie sólo pueden ser controladas por programa. El **+2A** no incluye software para el control de la puerta **AUX**, el cual deberá ser gestionado por el programa de usuario. En cambio, la puerta **RS232/MIDI** es controlada plenamente por +3 BASIC.

La manera en que trabaja el AY-3-8912 es bastante compleja; se recomienda a quienes se sientan tentados a experimentar que consulten la hoja de datos del AY-3-8912. No obstante, la siguiente información debería ser suficiente para empezar.

El circuito de sonido contiene dieciséis registros; para seleccionarlos, primero se escribe el número de registro en la puerta de escritura de direcciones, FFFDh (65533), y después se lee el valor del registro (en la misma dirección) o se escribe en la dirección de escritura de registros de datos, BFFDh (49149). Una vez seleccionado un registro, se puede realizar cualquier número de operaciones de lectura o escritura de datos. Sólo habrá que volver a escribir en la puerta de escritura de direcciones cuando se necesite seleccionar otro registro.

La frecuencia de reloj básica de este circuito es 1.7734 MHz (con precisión del 0.01%).

Los registros hacen lo siguiente:

R0 — Ajuste fino del tono, canal A
R1 — Ajuste aproximado del tono, canal A
R2 — Ajuste fino del tono, canal B
R3 — Ajuste aproximado del tono, canal B
R4 — Ajuste fino del tono, canal C
R5 — Ajuste aproximado del tono, canal C

El tono de cada canal es un valor de 12 bits que se forma combinando los bits D3-D0 del registro de ajuste aproximado y los bits D7-D0 del registro de ajuste fino. La unidad básica del tono es la frecuencia de reloj dividida por 16 (es decir, 110.83 KHz). Como el contador es de 12 bits, se puede generar frecuencias de 27 Hz a 110 KHz.

R6 — Control del generador de ruido, D4-D0

El periodo del generador de ruido se toma contando los cinco bits inferiores del registro de ruido cada periodo del reloj de sonido dividido por 16.

R7 — Control del mezclador y de E/S

D7 No utilizado
D6 1=puerta de entrada, 0=puerta de salida
D5 Ruido en el canal C
D4 Ruido en el canal B
D3 Ruido en el canal A
D2 Tono en el canal C
D1 Tono en el canal B
D0 Tono en el canal A

Este registro controla la mezcla de ruido y tono para cada canal y la dirección de la puerta de E/S de ocho bits. Un cero en un bit de mezcla indica que la función está activada.

R8 — Control de amplitud del canal A
R9 — Control de amplitud del canal B
RA — Control de amplitud del canal C

D4 1=utilizar generador de envolvente
0=utilizar el valor de D3-D0 como amplitud
D3-D0 Amplitud

Estos tres registros controlan la amplitud de cada canal y si ésta debe ser modulada o no por los registros de envolvente.

RB — Ajuste aproximado del periodo de envolvente
RC — Ajuste fino del periodo de envolvente

Los valores de ocho bits de RB y RC se combinan para producir un número de 16 bits que se cuenta en unidades de 256 por el periodo del reloj de sonido. Las frecuencias de envolvente pueden estar entre 0.1 Hz y 6 KHz.

RD — Control de envolventes

D3 Continua
D2 Ataque
D1 Alternada
D0 Sostenida

El diagrama de las formas de envolvente (Sección 19 de este capítulo) da una ilustración gráfica de los posibles estados de este registro.

Si está conectada la unidad de disquete externa, su control lo realiza el circuito controlador μPD765A del interfaz externo. Tal como explicamos en la Sección 23, el registro de datos de este dispositivo está en la dirección 3FFDh (16381); el registro de estado se encuentra en 2FFDh (12285). Este dispositivo es muy complejo, por lo que no se debe experimentar con él si no se conoce perfectamente su funcionamiento (consúltese la hoja de datos del fabricante).

La puerta de impresora paralelo (Centronics) es básicamente un 'latch' de 8 bits (74273) cuya dirección es 0FFDh (4093). La señal STROBE para la impresora es generada por la ULA, y está accesible en el bit 4 de la dirección 1FFDh (8189). El estado de la señal BUSY procedente de la impresora puede ser leído en el bit 0 de la dirección 0FFDh (4093).

Sección 31
BASIC

Temas tratados:

- Números
- Variables
- Cadenas
- Funciones
- Resumen de palabras clave
- Operaciones matemáticas

BASIC almacena los números con precisión de 9 o 10 dígitos. El mayor número que puede manejar BASIC es aproximadamente 10^{38}; el más pequeño (positivo) es aproximadamente 4×10^{-39}.

A menos que los números sean potencias exactas de 2, cabe la posibilidad de que las pequeñas imprecisiones internas lleguen a hacerse visibles al cabo de una sucesión de operaciones aritméticas. Esto es así en cualquier ordenador que no utilice la aritmética BCD ('decimal codificado en binario'). Se recomienda el trabajo con enteros en todos los casos en que se requiera precisión absoluta.

La forma de almacenamiento en el **+2A** es la *binaria de punto flotante*, con un byte para el exponente (e, $1\leqslant e\leqslant 255$) y cinco bytes para la mantisa (m, $½\leqslant m<1$), lo que representa el número $m\times2^{e-128}$.

Puesto que $½\leqslant m<1$, el bit más significativo de la mantisa m siempre es 1. Por lo tanto, podemos utilizarlo como indicador del signo (0 para los números positivos y 1 para los negativos).

Los enteros pequeños tienen una representación especial en la que el primer byte es 00h (0), el segundo es el byte del signo (00h o FFh) y el tercero y el cuarto son el entero propiamente dicho (en complemento a dos) con el byte menos significativo en primer lugar.

Los nombres de las variables numéricas son de longitud arbitraria; su primer carácter siempre es una letra y los siguientes pueden ser letras o dígitos. BASIC permite los espacios, pero los ignora; además convierte internamente todas las letras en minúsculas.

Los nombres de las variables de control de los bucles **FOR...NEXT** consisten en una sola letra.

Los nombres de las matrices numéricas consisten en una sola letra, que puede coincidir con el nombre de una variable sencilla. Estas matrices pueden tener múltiples dimensiones de tamaño arbitrario. Los subíndices empiezan en el 1.

Las cadenas son de longitud completamente arbitraria. Su nombre consiste en una sola letra seguida de **$**.

Las matrices literales pueden tener múltiples dimensiones de tamaño arbitrario. Su nombre consiste en una sola letra seguida de **$** y **no** puede coincidir con el nombre de una variable literal sencilla. Todas las cadenas de una matriz dada tienen la misma longitud fija, que está especificada por el último parámetro de la sentencia **DIM**. Los subíndices empiezan con el 1.

Disección de cadenas. Dada una cadena, se puede especificar una subcadena utilizando los llamados *cortadores*. Los cortadores pueden tener cualquiera de las siguientes formas:

(i) vacío
(ii) una expresión numérica
(iii) *expresión-numérica-opcional* **TO** *expresión-numérica-opcional*

La subcadena se expresa mediante:

(*a*) *expresión-literal***(***cortador***)**

o

(*b*) *variable-de-matriz-literal***(***subíndice***,**. . .**,***subíndice***,***cortador***)**

que es lo mismo que

*variable-de-matriz-literal***(***subíndice***,**. . .**,***subíndice***)(***cortador***)**

Supongamos que la *expresión-literal*, caso (*a*), es **s$**.

En el caso (*a*), si el cortador es vacío el resultado es la misma expresión literal **s$** (considerada como subcadena de sí misma).

Si el cortador es una expresión numérica, con valor *m*, el resultado es el *m*-ésimo carácter de **s$** (o sea, la subcadena tiene longitud 1).

Si el cortador tiene la forma (iii), supongamos que la primera expresión numérica tiene el valor *m* (el valor por defecto es 1) y la segunda el valor *n* (el valor por defecto es la longitud de la expresión literal). Si $1 \leqslant m \leqslant n \leqslant$ **LEN s$**, el resultado es la subcadena de **s$** que comienza en el *m*-ésimo carácter y acaba en el *n*-ésimo.

Si $0 \leqslant n < m$, el resultado es la cadena vacía. En cualquier otro caso se produce el error **3**.

BASIC realiza la disección de las cadenas antes de evaluar las funciones y operaciones, a menos que los paréntesis impongan lo contrario.

A las subcadenas se les puede asignar valor (véase **LET**). Si una cadena debe contener comillas, al especificarla se pone el signo de comillas (") repetido.

Funciones

El argumento de una función no necesita paréntesis si es una constante o una variable (opcionalmente, subindexada o producto de una disección).

Función	Tipo de argumento	Resultado
ABS	Número	Valor absoluto.
ACS	Número	Arco coseno en radianes. Error **A** si el argumento no está entre −1 y +1.
AND	Operación binaria. El operando de la derecha siempre es un número. Si el de la izquierda es un número: Si el de la izquierda es una cadena:	 **a AND b** es $\begin{cases} \mathbf{a} \text{ si } \mathbf{b}<>0 \\ \mathbf{0} \text{ si } \mathbf{b}=0 \end{cases}$ **a$ AND b** es $\begin{cases} \mathbf{a\$} \text{ si } \mathbf{b}<>0 \\ \text{""} \text{ si } \mathbf{b}=0 \end{cases}$ **AND** tiene prioridad de nivel 3.
ASN	Número	Arco seno en radianes. Error **A** si el argumento no está entre −1 y +1.
ATN	Número	Arco tangente en radianes.
ATTR	Dos argumentos, *x* e *y*, ambos numéricos, escritos entre paréntesis	Un número cuya forma binaria codifica los atributos de la posición de carácter que está en la línea *x*, columna *y*, de la pantalla. El bit 7 (el más significativo) es 1 para parpadeo, 0 para fijo. El bit 6 es 1 para brillo intenso, 0 para normal. Los bits 5 al 3 dan el color del papel. Los bits 2 al 0 dan el color de la tinta. Error **B** a menos que $0 \leqslant x \leqslant 23$ y $0 \leqslant y \leqslant 31$.
BIN		No es realmente una función, sino una notación alternativa para números: **BIN** seguido de una secuencia de ceros y unos es el número cuya representación binaria es la especificada por esa secuencia.

Función	Tipo de argumento	Resultado
CHR$	Número	El carácter cuyo código es el argumento (redondeado al entero más próximo).
CODE	Cadena	Código del primer carácter de la cadena (o 0, si se trata de la cadena vacía).
COS	Número (en radianes)	Coseno del argumento.
EXP	Número	Número *e* elevado al argumento.
FN		**FN**, seguida de una letra, invoca una función definida por el usuario (véase **DEF**). Los argumentos tienen que estar entre paréntesis. (Hay que poner los paréntesis aunque no haya argumentos.)
IN	Número	Resultado de leer (a nivel del microprocesador) la puerta de entrada especificada por el argumento x ($0 \leqslant x \leqslant$ FFFFh). Carga el par de registros BC con x y ejecuta la instrucción de lenguaje ensamblador **in a,(c)**.
INKEY$	Ninguno	Lee el teclado. El resultado es el carácter que representa la tecla pulsada, si está pulsada una y sólo una; de lo contrario, el resultado es la cadena vacía.
INT	Número	Parte entera (redondeando hacia abajo).
LEN	Cadena	Longitud del argumento.
LN	Número	Logaritmo neperiano o natural (de base *e*). Error **A** si $x \leqslant 0$.
NOT	Número	0 si $x <> 0$, 1 si $x = 0$. **NOT** tiene prioridad de nivel 4.
OR	Operación binaria. Ambos operandos son numéricos	**a OR b** es $\begin{cases} 1 \text{ si } \mathbf{b} <> 0 \\ \mathbf{a} \text{ si } \mathbf{b} = 0 \end{cases}$ **OR** tiene prioridad de nivel 2.
PEEK	Número	El valor del byte que está en la posición de memoria cuya dirección es el argumento (redondeado al entero más cercano). Error **B** si el argumento no está entre 0 y 65535.

Función	Tipo de argumento	Resultado
PI	Ninguno	Número π (3.1415927...).
POINT	Dos argumentos, x e y, ambos numéricos, escritos entre paréntesis	1 si el pixel especificado por (x,y) tiene el color de la tinta; 0 si tiene el del papel. Error **B** a menos que $0 \leqslant x \leqslant 255$ y $0 \leqslant y \leqslant 175$.
RND	Ninguno	El siguiente número pseudoaleatorio de una sucesión que se genera tomando módulo 65537 las potencias de 75, restando 1 y dividiendo por 65536. $0 \leqslant y < 1$.
SCREEN$	Dos argumentos, x e y, ambos numéricos, escritos entre paréntesis	El carácter que está (normal o invertido) en la intersección de la fila x con columna y. Da la cadena vacía si BASIC no reconoce el carácter. Error **B** a menos que $0 \leqslant x \leqslant 23$ y $0 \leqslant y \leqslant 31$.
SGN	Número	Signo del número. Da -1 para los negativos, 0 para el 0 y $+1$ para los positivos.
SIN	Número (en radianes)	Seno del argumento.
SQR	Número	Raíz cuadrada. Error **A** si el argumento es menor que 0.
STR$	Número	La cadena de caracteres que aparecería en la pantalla si se escribiera el número.
TAN	Número (en radianes)	Tangente del argumento.
USR	Número	Invoca la subrutina de código de máquina cuya dirección inicial es el argumento. A la entrada en la subrutina, la memoria está configurada de modo que 0000h...3FFFh está ocupado por ROM3 (la de 48 BASIC), 4000h...7FFFh está ocupado por la página 5 de la RAM, 8000h...BFFFh está ocupado por la página 2 de la RAM y C000h...FFFFh está ocupado por la página 0 de la RAM. Si se va a invocar rutinas de +3DOS, se debe encajar la página 7 de RAM en C000h...FFFFh y ROM2 (la de +3DOS) en 0000h...3FFFh. (Véase la Sección 26 de este capítulo.) Al retornar de la rutina, el resultado es el contenido del par de registros BC.

Función	Tipo de argumento	Resultado
	Cadena	Dirección de la descripción de la forma del gráfico de usuario correspondiente al argumento. Error **A** si el argumento no es una sola letra, de la **a** a la **u**, o un gráfico de usuario.
VAL	Cadena	Evalúa el argumento (tras suprimir las comillas) como si fuera una expresión numérica. Error **C** si el argumento no es válido como expresión numérica. Pueden presentarse otros errores, dependiendo de la forma de la expresión.
VAL$	Cadena	Evalúa el argumento (tras suprimir las comillas de sus extremos) y da la expresión literal resultante. Error **C** si el argumento contiene un error de sintaxis o da un valor numérico. Pueden presentarse otros errores, dependiendo de la expresión.
–	Número	Negación.

Operaciones binarias:

+	Suma de números o concatenación de cadenas
–	Resta
*	Multiplicación
/	División
↑	Elevación a una potencia. Error **B** si el operando de la izquierda es negativo
=	Igual a
>	Mayor que
<	Menor que
<=	Igual o menor que
>=	Igual o mayor que
<>	Distinto de

(Para =, >, <, <=, >=, <>:) Ambos operandos deben ser del mismo tipo. El resultado es el número 1 si la proposición es 'verdadera'; 0 si es 'falsa'.

Las funciones y operaciones tienen las siguientes prioridades:

Operación	Nivel de prioridad
Subíndices y disección de cadenas	12
Todas las funciones excepto **NOT** y negación	11
↑	10
Negación (signo − usado para negar)	9
*, /	8
=, − (signo − usado para restar)	6
=, >, <, <=, >=, <>	5
NOT	4
AND	3
OR	2

Sentencias

Notación utilizada en esta lista:

a Representa una sola letra.
v Representa una variable.
x, *y*, *z* Representan expresiones numéricas.
m, *n* Representan expresiones numéricas que BASIC redondea al entero más próximo.
e Representa una expresión.
f Representa una expresión cuyo valor es una cadena.
s Representa una secuencia de sentencias separadas por signos de dos puntos.
c Representa una secuencia de cláusulas de color, cada una de las cuales termina en una coma o en un signo de punto y coma. Una cláusula de color tiene la forma de una sentencia **PAPER, INK, FLASH, BRIGHT, INVERSE** u **OVER**.
u Representa una expresión literal cuyo valor es una denominación de unidad: **A**:, **B**:, **M**: o **T**:.
p Representa una especificación de fichero (para DOS) que puede ser ambigua, es decir, que puede contener los caracteres polivalentes **?** y *.
q Representa una especificación de fichero que no puede ser ambigua.

Las expresiones opcionales están escritas entre corchetes *[ ]*.

En todo lugar en que se requiera una constante se puede poner también una expresión (excepto para el número de línea al principio de una sentencia).

Todas las sentencias, a excepción de **INPUT**, **DEF FN** y **DATA**, pueden ser usadas como órdenes directas o en líneas de programa (aunque no tendrán la misma utilidad en los dos casos). Tanto las órdenes directas como las líneas de programa pueden constar de varias sentencias separadas por signos de dos puntos. No hay restricción alguna sobre en qué lugar de la línea puede aparecer una sentencia concreta; véase, no obstante, **IF** y **REM**.

BEEP *x,y* — Hace sonar una nota a través del altavoz del televisor durante *x* segundos a una altura de *y* semitonos por encima de la nota DO medio (o por debajo, si *y* es negativo).

BORDER *m* — Establece el color del borde de la pantalla superior y también el color del papel para la pantalla inferior.

Error **K** a menos que 0<=*m*<=7 (o sea, cuando *m* no esté en el margen de 0 a 7).

BRIGHT *m* — Establece el brillo de los caracteres que se escriba en lo sucesivo; *m*=0 para normal, 1 para brillo y 8 para transparente.

Error **K** si *m* no es 0, 1 ni 8.

CAT *[#n,][u][p]* — Produce un catálogo (en orden alfanumérico) de los ficheros de un disquete (si está conectada la unidad externa) o del disco de RAM. Si se incluye la opción *#n*, la salida se dirige al canal *n*. Si se incluye un nombre de fichero o una plantilla, el catálogo contiene solamente los ficheros que encajan en la especificación. Si sólo se especifica la unidad, el catálogo contiene todos los ficheros del disco. Si la unidad es **T:**, se obtiene la lista de los ficheros de la cinta (junto con información que será útil para transferir los ficheros de cinta a disco).

CAT *[#n,][u][p]* **EXP** — Es similar a **CAT**, pero produce un catálogo ampliado que incluye los ficheros de sistema e indica qué ficheros tienen activados los atributos de 'protección', 'sistema' o 'archivo' (véase **MOVE** *q* **TO** *f*).

CIRCLE *x,y,z* — Dibuja un arco de circunferencia, con centro en (*x*,*y*) y radio *z*.

CLEAR — Borra todas las variables, liberando así el espacio que ocupaban. Ejecuta **RESTORE** y **CLS**; reajusta la posición del cursor gráfico en el extremo inferior izquierdo y borra la pila de **GO SUB**.

CLEAR *n* — Igual que **CLEAR**, pero cambia la variable de sistema RAMTOP a *n* (si ello es posible) y coloca en esa dirección la nueva pila de **GO SUB**.

Se puede usar esta orden para colocar la pila de máquina por debajo de BFE0h (49120) antes de invocar desde BASIC una rutina que vaya a usar rutinas de +3DOS.

CLOSE *#n* — Deshace la conexión entre el canal número *n* y el dispositivo al que estaba asignado. El número queda así disponible para ser usado en una orden **OPEN** *#n,f* posterior.

CLS — Borra el fichero de imagen (memoria de pantalla).

CONTINUE — Reanuda la ejecución del programa a partir de la sentencia en que éste se había detenido emitiendo un informe distinto del **0**. Si el mensaje fue **9** o **L**, continúa a partir de la sentencia siguiente; de lo contrario, repite la sentencia en la que se produjo el error. Si el último informe se produjo en una línea de órdenes, **CONTINUE** intentará completar dicha línea y entrará en un bucle si el error fue en **0:1**, generará el mensaje **0** si fue en **0:2**, o bien el mensaje **N** si tuvo lugar en **0:3** o posterior.

COPY — Envía a la impresora (si está conectada) una copia de las 22 líneas superiores de la pantalla en formato de mapa de bits Epson de densidad cuádruple. Si se pulsa [BREAK], BASIC emite el mensaje **D** (y la impresora puede quedar en modo gráfico, con una interlínea inadecuada para la impresión de texto).

COPY EXP *[***INVERSE***]* — Envía a la impresora (si está conectada) una copia de las 24 líneas de la pantalla en formato de mapa de bits Epson de densidad cuádruple. A cada punto de color de la pantalla se hace corresponder en el papel una trama de puntos diferente; de esta forma se obtiene una gama de grises para representar los diversos colores y los valores de **BRIGHT**. La opción **INVERSE** invierte el resultado (como en un negativo fotográfico). Si se pulsa [BREAK], BASIC emite el mensaje **D** (y la impresora puede quedar en modo gráfico, con una interlínea inadecuada para la impresión de texto).

COPY q_1 **TO** q_2
COPY *p* **TO** *u*
COPY *u* **TO** *u* — (Órdenes de disco.) Copia el primer fichero hacia el segundo. Las especificaciones de fichero tienen que ser diferentes; pueden incluir letras de unidad y números de usuario.

Si la especificación de origen es ambigua (plantilla construida con caracteres polivalentes), la de destino tiene que ser solamente una letra de unidad. (En este caso los ficheros de destino tendrán los mismos nombres y tipos que los de origen.)

Si ambas especificaciones consisten en una letra de unidad, la orden realiza una transferencia completa de un disco a otro (y borra los ficheros que hubiera antes en el disco de destino). Esta transferencia no funciona si el disco de destino no tiene el formato del **+3** (que es el formato utilizado por la unidad externa del **+2A**).

Al copiar ficheros individuales, la orden produce el mensaje '**YA EXISTE EL FICHERO**' en caso de que en el disco haya un fichero con el nombre de destino. El mensaje '**SIN MARCA DIRECCIONES**' significa normalmente que el disco de destino no está formateado.

COPY *q* **TO SCREEN$** Exhibe en la pantalla el contenido de un fichero de disco, reemplazando los caracteres de control (tabuladores, avances de línea, etc.) con espacios. Esta orden no es adecuada para examinar ficheros de programa de BASIC, sino sólo para ficheros de texto ASCII.

COPY *q* **TO LPRINT** Envía a la impresora el contenido del fichero, sin realizar ninguna conversión de caracteres. Si previamente se ha ejecutado la orden **FORMAT LPRINT "R"** para desviar la salida de impresora hacia al puerta serie, esta orden permite transferir programas a otra máquina.

COPY *q* **TO SPECTRUM FORMAT** Añade una cabecera a un fichero binario. El resultado es un nuevo fichero con el mismo nombre y con el tipo **HED**.

DATA $e_1, e_2, e_3, \ldots$ Define un tramo de la lista de datos. Debe estar en un programa, pues en una orden directa no produciría ningún efecto.

DEF FN $a(a_1, \ldots, a_k)=e$ Define una función de usuario. Debe estar en un programa, pues en una orden directa no produciría ningún efecto.

a y $a_1 \ldots a_k$ son, cada una, una sola letra (o una letra seguida de **$** para especificar un resultado o un argumento literal).

Toma la forma **DEF FN** *a()=e* si la función no lleva argumentos.

DIM $a(n_1, \ldots, n_k)$ Borra la matriz *a* (si existe) y forma la matriz numérica *a* con *k* dimensiones, $n1, \ldots, n_k$. Inicializa todos los valores a 0.

DIM $a\$(n_1, \ldots, n_k)$ Borra la matriz o cadena *a$* (si existen) y forma una matriz de caracteres *a$* con *k* dimensiones, $n_1, \ldots, n_k$. Inicializa todos los valores a "". Se puede considerar esta matriz como matriz de cadenas de longitud fija n_k, con $k-1$ dimensiones $(n_1, \ldots, n_k-1)$. Una matriz está indefinida hasta que se la dimensiona con **DIM**.

Error **4** si no queda espacio para la matriz en la memoria.

DRAW *x,y* Equivale a **DRAW** *x,y*,**0**.

DRAW *x,y,z* Dibuja una línea (recta o arco de circunferencia) desde la posición actual del cursor gráfico desplazándose *x* unidades en horizontal

e *y* unidades en vertical en relación con el punto de partida, y al mismo tiempo girando un ángulo *z*.

Error **B** si la línea se sale de la pantalla.

ERASE *p*
ERASE *u*

(Órdenes de disco.) Si sólo se especifica un fichero, borra ese fichero en la unidad especificada (o en la implícita, si no se especifica ninguna). Si se especifica una plantilla, aparece un mensaje para pedir confirmación; si entonces se responde pulsando [S], la orden borra todos los ficheros que encajan en la plantilla. Si la especificación consiste en una letra de unidad, la orden borra todos los ficheros de la unidad sin pedir confirmación.

FLASH *n*

Establece si los caracteres deben ser parpadeantes o fijos; *n*=0 para fijos, *n*=1 para parpadeo, *n*=8 para que no haya cambios.

FOR *a*=*x* **TO** *y* **FOR** *a*=*x* **TO** *y* **STEP 1** **FOR** *a*=*x* **TO** *y* **STEP** *z*

Borra la variable ordinaria *a* (si existe); define la variable de control *a* con valor inicial *x*, límite *y*, paso *z* y con una dirección de retorno del bucle que hace referencia a la sentencia que sigue a **FOR**. Comprueba si el valor inicial es mayor (si $z \geqslant 0$) o menor (si $z<0$) que el límite; en caso afirmativo, salta a la sentencia **NEXT** *a*, dando el error **1** si ésta no existe. Véase **NEXT**.

Error **4** si no queda espacio para la variable de control en la memoria.

FORMAT *u*

Prepara el disco de la unidad especificada (**A**: o **B**:) para que el ordenador pueda escribir en él. Si el disco ya ha sido formateado en el **+2A** (o en el **+3**), aparece un mensaje que da la oportunidad de abandonar la operación. Este ordenador normalmente no reconocerá los discos formateados en otras máquinas (salvo los del AMSTRAD PCW, CF-2). Esta orden genera un error si no está conectada la unidad de disquete externa.

FORMAT LINE *n*

Establece en *n* baudios la velocidad de transmisión para el interfaz serie RS232. Velocidades válidas son las comprendidas entre 75 y 19200.

FORMAT LPRINT $f_1[;f_2]$

Dirige la salida de impresora hacia la puerta serie o la puerta paralelo. Decide si los caracteres enviados a la impresora deben ser transformados.

Si la cadena f_1 es **"C"**, la salida de impresora se dirige hacia la puerta paralelo Centronics (zócalo **IMPRESORA**). Si es **"R"**, la salida va a la puerta serie (zócalo **RS232**).

f_1 también puede ser **"E"**; en tal caso los caracteres de código inferior al **32** no son enviados a la impresora, y los códigos de palabras clave de BASIC (superiores al **127**) son convertidos en caracteres de texto. Cuando f_1 es **"U"**, todos los caracteres son enviados a la impresora sin conversión alguna; esto permite el uso de secuencias de escape para controlar la impresora.

Si f_1 es **"C"** o **"R"**, se puede incluir un segundo parámetro, f_2, que puede ser **"E"** o **"U"**, con el significado antes descrito.

GO SUB *n* — Deposita el número de línea *n* en la pila; después actúa como **GO TO** *n*.

Error **4** si no hay suficientes sentencias **RETURN**.

GO TO *n* — Salta a la línea de número *n* (o, si no existe, a la primera línea posterior a ese número).

IF *x* **THEN** *s* — Si *x* es 'verdadero' (distinto de cero), ejecuta *s*. Nótese que *s* comprende todas las sentencias hasta el final de la línea. La forma **IF** *x* **THEN** *número-de-línea* no está permitida en +3 BASIC.

INK *n* — Establece el color de la tinta (primer plano) para los caracteres que se escriba en lo sucesivo; *n* está en el margen de 0 a 7 para un color, *n*=8 para transparentes y *n*=9 para contraste.

Error **K** a menos que $0 \leqslant n \leqslant 9$.

INPUT *[#n,]...* — '...' es una secuencia de elementos de **INPUT**, separados, al igual que en una sentencia **PRINT**, por comas, signos de punto y coma o apóstrofos. Un elemento de **INPUT** puede ser cualquiera de los siguientes:

(i) Cualquier elemento de **PRINT** que no empiece por una letra.
(ii) Un nombre de variable.
(iii) **LINE** seguido de un nombre de variable de tipo literal.

Los elementos de **PRINT** y los separadores del apartado (i) son tratados exactamente igual que en **PRINT**, con la excepción de que la escritura se dirige a la pantalla inferior. Para (ii), el ordenador se detiene y espera la introducción de una expresión por el teclado; el valor de esa expresión es asignado a la variable. Los caracteres escritos son reproducidos en la pantalla en la forma habitual; los errores de sintaxis producen un signo de interrogación en negativo y parpadeante. En el caso de expresiones de tipo literal, el tampón de entrada se inicializa de forma que contenga dos comillas (que el usuario puede borrar, si es necesario). Si el primer carácter de la entrada es **STOP** ([SIMB] [A]), el programa se detiene y emite el mensaje **H**. (iii) es igual que (ii), con la diferencia de que la entra-

	da es tratada como constante literal sin comillas y el mecanismo de **STOP** no funciona (en su lugar, para detener la sentencia hay que pulsar la tecla [↓]).
INVERSE *n*	Controla la inversión de los caracteres que se escriba en lo sucesivo. Si *n*=0, los caracteres aparecen en video normal, es decir, color de la tinta sobre color del papel. Si *n*=1, los caracteres aparecen en video inverso, esto es, color del papel sobre color de la tinta. Error **K** si *n* no es 0 ni 1. En 48 BASIC, la pulsación de la tecla [VIDEO INV] es equivalente a **INVERSE 1**; la pulsación de [VIDEO NORM] equivale a **INVERSE 0**.
LET *v=e*	Asigna el valor de *e* a la variable *v*. No se puede omitir la palabra 'LET'. Una variable sencilla está indefinida mientras no se le asigne valor en una sentencia **LET**, **READ** o **INPUT**. Si *v* es una variable literal indexada o una variable literal obtenida por disección de cadenas (subcadena), la asignación es «procrusteana» (de longitud fija); es decir, el valor de *e* es truncado o rellenado con espacios por la derecha, con el fin de darle la longitud que se ha especificado para *v*.
LIST *[#m]*	Equivale a **LIST** *[#m,]***0**.
LIST *[#m,]n*	Lista el programa en la pantalla superior, comenzando en la primera línea cuyo número es, como mínimo, *n*. Convierte la línea *n* en línea actual. Si se incluye la opción *#m*, el listado es enviado al dispositivo asociado al canal número *m*.
LLIST	Equivale a **LLIST 0**.
LLIST *n*	Como **LIST**, pero enviando el listado a la impresora. Inicialmente toda la salida de impresora se dirige a la puerta paralelo (Centronics), pero se la puede desviar a la puerta serie RS232 ejecutando una orden **FORMAT LPRINT "R"**. A fin de que los programas de BASIC queden correctamente listados, **LLIST** convierte los códigos de palabras clave en caracteres de texto, y no envía los códigos inferiores al **32**. Para impedir esta transformación se ejecuta una orden **FORMAT LPRINT "U"**; para restablecerla se ejecuta **FORMAT LPRINT "E"**.
LOAD *u*	Selecciona como nueva unidad implícita la especificada por *u*. La unidad implícita es la que será utilizada por defecto en lo sucesivo en las órdenes de disco (**COPY**, **ERASE**, **MOVE**, etc.). Si *u* es **T:**, todas las órdenes **LOAD** posteriores tratarán de leer en la cinta.

LOAD *f* — Carga el programa y las variables a partir de la cinta (o del disco). *f* puede incluir la letra de unidad y el número de usuario. Si no se especifica la unidad, **LOAD** utiliza la implícita.

Si *f* es **"*"**, **LOAD** trata de cargar el sector de inicialización del disco. De esta forma se puede cargar un programa de juego o un sistema operativo.

LOAD *f* **DATA** *a***()** — Carga la matriz numérica *a***()** leyéndola en el fichero *f*.

LOAD *f* **DATA** *a***$()** — Carga la matriz literal *a***$()** leyéndola en el fichero *f*.

LOAD *f* **CODE** *m,n* — Carga (como máximo) *n* bytes, comenzando en la dirección *m*.

LOAD *f* **CODE** *m* — Carga bytes comenzando en la dirección *m*. Si *f* es un fichero creado en otra máquina y convertido al formato de Spectrum (con la orden **COPY** *q* **TO SPECTRUM FORMAT**), ésta es la forma de **LOAD** con la que hay que cargarlo, ya que la cabecera no contendrá la dirección de carga.

LOAD *f* **CODE** — Carga bytes en la dirección desde la cual fueron grabados.

LOAD *f* **SCREEN$** — Equivale a **LOAD** *f* **CODE 16384,6912**

LPRINT . . . — Como **PRINT**, pero enviando la salida hacia la impresora. Inicialmente toda la salida de impresora se dirige a la puerta paralelo (Centronics), pero se la puede desviar a la puerta serie RS232 ejecutando una orden **FORMAT LPRINT "R"**; entonces se puede usar **LPRINT** para enviar cadenas de caracteres a un ordenador o terminal remoto.

Por defecto, **LPRINT** convierte los códigos de palabras clave de BASIC en caracteres de texto, y no envía los códigos inferiores al **32**. Para impedir esta transformación se ejecuta una orden **FORMAT LPRINT "U"**; para restablecerla se ejecuta **FORMAT LPRINT "E"**.

MERGE *f* — Como **LOAD** *f*, pero no borra las líneas ni las variables del programa antiguo, salvo las que tengan el mismo número o nombre que en el programa nuevo. Al igual que en **LOAD**, *f* puede incluir la letra de unidad y el número de usuario. Si no se especifica la unidad, **MERGE** utiliza la implícita.

MOVE f_1 **TO** f_2 — (Orden de disco.) Da al fichero *f*1 el nuevo nombre *f*2. Ambas especificaciones de fichero tienen que hacer referencia (explícita o implícita) a la misma unidad.

MOVE *q* **TO** *f* — (Orden de disco.) La cadena *f* puede ser **"+P"**, **"+S"**, **"+A"**, **"−P"**, **"−S"** o **"−A"**. Esta orden activa (+) o desactiva (−) los

atributos de 'protección contra escritura' (**P**), 'sistema' (**S**) y 'archivo' (**A**) del fichero *q*. Se puede ejecutar una orden **CAT ... EXP** para averiguar la situación actual de los atributos. Los ficheros protegidos no pueden ser borrados ni sustituidos por versiones más modernas, ni se les puede aplicar ninguna operación que los modifique. Los ficheros de 'sistema' no son visibles para **CAT** (pero sí para **CAT EXP**). El atributo de 'archivo' ha sido incluido por razones de compatibilidad con CP/M, pero no tiene utilidad en el **+2A**.

NEW Inicializa el sistema BASIC, borrando el programa (y sus variables) y toda la memoria a la que BASIC tiene acceso (hasta la dirección dada por la variable de sistema RAMTOP). Conserva las variables de sistema UDG, P_RAMT, RASP y PIP. Devuelve el control al menú de presentación, pero no afecta al disco de RAM (unidad M).

NEXT *a*

(i) Busca la variable de control *a*.

(ii) Añade a su valor el valor del paso especificado en **FOR**.

(iii) Si el paso es igual o mayor que 0 y el valor de *a* es mayor que el límite (o si el paso es menor que 0 y el valor de *a* es menor que el límite), salta a la sentencia de retorno del bucle.

Error **2** si no existe la variable *a*.

Error **1** si la variable *a* no es la misma que la especificada en **FOR**.

OPEN *#n,f* Asocia al canal número *n* el dispositivo especificado por *f*. Los números de canal pueden estar en el margen de 0 a 15; no obstante, el sistema se reserva los cuatro primeros, 0...3, por lo que no es conveniente utilizarlos. Los dispositivos posibles son **"S"** (pantalla), **"K"** (teclado) y **"P"** impresora. La impresora puede ser redirigida a la puerta paralelo o a la puerta serie (con **FORMAT LPRINT**). Se provoca el error '**DISP. E/S INCORRECTO**' si se intenta leer en un dispositivo que sólo admite escritura, o viceversa.

OUT *m,n* Envía el byte *n* a la puerta *m*. [Carga el par de registros BC con *m* y el registro A (acumulador) con *n* y ejecuta la instrucción **out (c),a.**]

Error **B** a menos que $0 \leqslant m \leqslant 65535$ y $-255 \leqslant n \leqslant 255$.

OVER *n* Controla la sobreimpresión para los caracteres que se escriba en lo sucesivo.

Si $n=0$, los caracteres sustituyen a los que hubiera previamente en la posición de escritura.

Si $n=1$, los caracteres nuevos se mezclan con los antiguos para dar el color de la tinta dondequiera que uno de ellos (pero no ambos) tuviese dicho color, y el color del papel donde ambos fuesen papel o ambos tinta.

Error **K** a menos que *n* sea 0 o 1.

PAPER *n* — Como **INK**, pero para controlar el color del papel (fondo).

PAUSE *n* — Detiene el programa y muestra la imagen durante *n* barridos del cuadro (a 50 barridos por segundo), o hasta que se pulsa una tecla. Si $n=0$ la pausa dura hasta que se pulsa una tecla.

Error **B** a menos que $0 \leqslant n \leqslant 65535$.

PLAY $f_1[,f_2,\ldots f_8]$ — Interpreta hasta ocho cadenas (véase la Sección 19 de este capítulo) y las ejecuta simultáneamente. Las primeras tres cadenas son ejecutadas a través del altavoz del televisor y (opcionalmente) a través de la puerta de MIDI; todas las siguientes van a la puerta MIDI.

PLOT *c;m,n* — Dibuja un punto de tinta (supeditado a **OVER** y a **INVERSE**) en el pixel (*m*,*n*); establece en este punto la nueva 'posición actual' del cursor gráfico.

A no ser que los elementos de color *c* especifiquen otra cosa, el color de la tinta de la posición de carácter que contiene el pixel se cambia por el color de tinta permanente actual, y los otros (color del papel, parpadeo y brillo) permanecen como estaban.

Error **B** a menos que $0 \leqslant m \leqslant 255$ y $0 \leqslant n \leqslant 175$.

POKE *m,n* — Escribe el valor *n* en la posición de memoria cuya dirección es *m*.

Error **B** a menos que $0 \leqslant m \leqslant 65535$ y $-255 \leqslant n \leqslant 255$.

PRINT *[#n,]*. . . — '. . .' es una secuencia de elementos de **PRINT**, separados entre sí por comas, signos de punto y coma o apóstrofos. **PRINT** los escribe en el fichero de imagen (memoria de pantalla) para su salida por la pantalla.

Si se incluye la opción *#n*, la salida se dirige al dispositivo asociado al canal número *n* (que también puede ser la pantalla, si se especificó **"S"** al abrir el canal).

Un punto y coma entre dos elementos no produce ningún efecto (sirve sólo para separarlos); una coma provoca el avance hasta el principio de la siguiente zona de escritura; un apóstrofo produce el 'retorno del carro' (equivalente a la pulsación de la tecla [**INTRO**]), que **PRINT** emite por defecto si la sentencia no termina en punto y coma, coma o apóstrofo).

Un elemento de **PRINT** puede ser:

(i) Vacío; es decir, nada.

(ii) Una expresión numérica. **PRINT** escribe primero un signo menos si el valor es negativo. Supongamos que x es el valor absoluto de la expresión. Si $x \leqslant 10^{-5}$ o $x \geqslant 10^{13}$, el número se escribe en notación científica. La mantisa tiene hasta ocho dígitos (sin ceros por la derecha); el punto decimal (ausente si sólo hay un dígito) va detrás del primero. La parte del exponente es **E**, seguida de + o − y, a continuación, de uno o dos dígitos. Si x no está en el margen mencionado, el número se escribe en notación decimal con hasta ocho dígitos significativos y sin ceros por la derecha después del punto decimal.

(iii) Una expresión literal. Los códigos de palabras clave de BASIC son transformados en caracteres de texto, posiblemente con un espacio antes o después. Los caracteres de control producen su efecto de control. Los caracteres irreconocibles se convierten en **?**.

(iv) **AT** *m,n*. Emite un carácter de control **AT** seguido por un byte para *m* (número de fila) y otro para *n* (número de columna).

(v) **TAB** *n*. Emite un carácter de control **TAB** seguido por dos bytes para *n* (primero el byte menos significativo), que es tope de tabulación.

(vi) Un elemento de color, que toma la forma de una sentencia **PAPER, INK, FLASH, BRIGHT, INVERSE** u **OVER**.

RANDOMIZE — Equivale a **RANDOMIZE 0**.

RANDOMIZE *n* — Inicializa la variable de sistema SEED que se va a usar para generar el próximo valor de **RND**. Si $n<>0$, se le da a SEED el valor n. Si $n=0$, SEED toma el valor de otra variable de sistema, FRAMES, que lleva la cuenta de los barridos del cuadro exhibidos hasta ahora en la pantalla y que, por consiguiente, debería ser bastante aleatoria.

Error **B** a menos que $0 \leqslant n \leqslant 65535$.

READ $v_1, v_2, \ldots v_k$ — Asigna valores a las variables, leyéndolos en las sucesivas expresiones de la lista de datos.

Error **C** si una expresión es de tipo inadecuado para la variable a la que debería ser asignada.

Error **E** si se ha agotado la lista de datos cuando todavía quedaban variables a las que asignar valor.

REM ...	No produce ningún efecto. '...' puede ser cualquier secuencia de caracteres que termine en 'retorno del carro' (**INTRO**). No se ejecutará ninguna sentencia que esté detrás de la palabra 'REM' en la misma línea; los signos de dos puntos **no** serán considerados separadores de sentencias.
RESTORE	Equivale a **RESTORE 0**.
RESTORE *n*	Dirige el «puntero» de datos hacia la primera sentencia **DATA** de la línea *n*. Si no existe esa línea (o si no contiene ninguna sentencia **DATA**), el puntero se dirige hacia la primera sentencia **DATA** posterior a la línea *n*. La siguiente sentencia **READ** empezará a leer a partir de ese punto.
RETURN	Lee en la pila de **GO SUB** la referencia a una sentencia y salta a la línea siguiente a ella. Error **7** cuando en la pila no hay referencia a ninguna sentencia. (Esto generalmente indica que los **GO SUB** no están bien emparejados con los **RETURN**.)
RUN	Equivale a **RUN 0**.
RUN *n*	Ejecuta **CLEAR** y después **GO TO** *n*.
SAVE *u*	Selecciona como nueva unidad implícita la especificada por *u*. La unidad implícita es la que será utilizada por defecto en lo sucesivo en las órdenes de disco (**COPY, ERASE, MOVE,** etc.). Si *u* es **T:**, todas las órdenes **SAVE** posteriores escribirán en la cinta.
SAVE *f*	Graba el programa y las variables en un fichero (de disco o de cinta) al que da el nombre *f*. Si la grabación se realiza en disco, *f* puede incluir la letra de unidad y el número de usuario. Si no se especifica la unidad, **SAVE** utiliza la implícita. Error **F** si el nombre está vacío o consta de más de diez caracteres (para grabación en cinta).
SAVE *f* **LINE** *m*	Graba el programa y las variables de forma que, cuando más tarde se lo cargue, BASIC realizará un salto automático a la línea *m*.
SAVE *f* **DATA** *a*()	Graba la matriz numérica *a*() en el fichero *f*.
SAVE *f* **DATA** *a$*()	Graba la matriz literal *a$*() en el fichero *f*.
SAVE *f* **CODE** *m,n*	Graba *n* bytes comenzando en la dirección *m*.
SAVE *f* **SCREEN$**	Equivale a **SAVE** *f* **CODE 16384,6912**. Graba la imagen de pantalla actual.

SPECTRUM Pasa de +3 BASIC a 48 BASIC, conservando el programa que pueda haber en la RAM. De 48 BASIC no se puede volver a 128 BASIC. La conmutación de ROM y RAM no se inhibe al entrar en 48 BASIC de esta forma (a diferencia de lo que ocurre cuando se selecciona **48 BASIC** en el menú de presentación).

STOP Detiene el programa y emite el informe **9**. **CONTINUE** reanudará la ejecución del programa a partir de la sentencia siguiente.

VERIFY *f* Como LOAD (aplicada a la cinta), pero sin cargar en la RAM la información leída, sino sólo comparándola con la que hay actualmente en la RAM.

No produce ningún efecto si *f* especifica un fichero de disco, o si la unidad implícita es A, B o M.

Error **R** si la comparación detecta alguna diferencia entre lo grabado en la cinta y lo almacenado en la RAM.

Sección 32
Binario y hexadecimal

Temas tratados:

Sistemas de numeración
Bits y bytes

Esta sección describe la forma en la que los ordenadores manejan internamente los números, utilizando el sistema binario.

En la mayor parte de los idiomas europeos, los nombres de los números reflejan una relación fundamental con el número diez. En castellano, por ejemplo, solamente son excepcionales, en este sentido, el 'once', el 'doce', el 'trece', el 'catorce' y el 'quince'. A partir de éste la regularidad es manifiesta:

... dieciséis, diecisiete, dieciocho, diecinueve
veinte, veintiuno, veintidós, ..., veintinueve
treinta, treinta y uno, treinta y dos, ..., treinta y nueve
cuarenta, cuarenta y uno, cuarenta y dos, ..., cuarenta y nueve
...

Esto facilita el uso sistemático de los números. La razón por la que el número diez desempeña este papel fundamental es el hecho de que tenemos diez dedos en las manos. Este sistema de numeración se llama *decimal*.

Los ordenadores, en lugar del sistema decimal (de *base* diez), utilizan el *hexadecimal* (abreviadamente, hex), cuya base es el número dieciséis. Ya que en nuestro sistema numérico decimal sólo tenemos diez dígitos, necesitamos otros seis para representar los números en hexadecimal. Las seis cifras adicionales son A, B, C, D, E y F. ¿Y qué viene después de la F? Bien, cuando se nos acaban los dígitos en el sistema decimal, volvemos a empezar por el 0 poniéndole un 1 delante (10); lo mismo ocurre en el sistema hexadecimal: el número que sigue a F es 10 (que es el 16 decimal), el siguiente es 11 (17 decimal), etc.

Veamos una tabla que da la equivalencia entre los dos sistemas:

Decimal	Hexadecimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	10
17	11
:	:
25	19
26	1A
27	1B
:	:
31	1F
32	20
33	21
:	:
158	9E
159	9F
160	A0
161	A1
:	:
255	FF
256	100

etcétera.

Si un número está escrito en notación hexadecimal, se puede evidenciar este hecho escribiendo una 'h' como sufijo. Por ejemplo, el número 256 decimal es 100h en hexadecimal.

Puede usted estar preguntándose qué tiene que ver todo esto con los ordenadores. De hecho, los ordenadores se comportan como si sólo tuvieran dos dígitos, representados por una tensión eléctrica baja ('apagado', 0) y una tensión alta ('encendido', 1). Este sistema de numeración en que sólo se dispone de dos dígitos se denomina *binario*. Los dos dígitos se llaman *bits* (de *bi*nary digi*t*, dígito binario). Así pues, un bit es un estado que puede tener dos valores: 0 o 1.

La siguiente tabla da las versiones decimal, hexadecimal y binaria de los primeros números:

Decimal	Hex	Binario
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111
16	10	10000
17	11	10001
	etcétera.	

Es habitual rellenar los números binarios con ceros por la izquierda, de forma que siempre contengan al menos cuatro bits; por ejemplo, 0000, 0001, 0010, 0011 (que representan los números 0 al 3 en decimal).

La conversión entre binario y hexadecimal es muy fácil (consultando la tabla anterior):

Para convertir un número binario en hexadecimal, divida el binario en grupos de cuatro bits (comenzando por la derecha del número) y convierta cada grupo en su correspondiente dígito hexadecimal. Finalmente, una los dígitos hexadecimales para formar el número hex completo. Por ejemplo, para pasar el número binario 10110100 a hexadecimal, convierta el primer grupo (por la derecha) de cuatro bits (0100) en el hexadecimal 4; después convierta el siguiente grupo (1011) en el hexadecimal B, únalos y obtendrá el número hexadecimal completo, B4h. Si el número binario tiene más de ocho bits, puede continuar convirtiendo cada grupo de cuatro bits en un dígito hex. Por ejemplo, el número binario 11101011110000 corresponde al 3AF0h.

Para convertir un número hexadecimal en binario, transforme cada dígito hex en cuatro bits y a luego una los bits para formar un número binario completo. Por ejemplo, para pasar F3h a binario, convierta primero a 3 en su correspondiente 0011 binario (recuerde que debe rellenar con ceros por la izquierda para que el número binario conste de cuatro bits); después transforme F en su correspondiente 1111 binario, únalos y tendrá el número binario completo, 11110011.

A pesar de que los ordenadores usan un sistema binario puro, los humanos a menudo escribimos los números que vamos a suministrar al ordenador utilizando la notación hexadecimal; después de todo, el número 3AF0h, por ejemplo, es más fácil de leer y recordar que su equivalente, 0011101011110000, en notación binaria de dieciséis bits.

Los bits que se encuentran dentro del ordenador están agrupados normalmente en bloques de ocho, es decir, en *bytes*. Un byte puede representar cualquier número decimal del 0 al 255 (11111111 binario o FFh).

Se puede agrupar dos bytes para formar lo que se llama técnicamente una *palabra*. Una palabra se puede expresar mediante dieciséis bits o cuatro dígitos hex y representa un número decimal del 0 al 65535 (1111111111111111 binario o FFFFh).

Un byte se compone *siempre* de ocho bits, pero la longitud de las palabras varía de un ordenador a otro.

La notación BIN usada en la Sección 14 de este capítulo proporciona un medio de introducir números binarios en el **+2A**. Por ejemplo, **BIN 10** representa el 4 decimal, **BIN 111** representa el 7 decimal, **BIN 11111111** representa el 255 decimal, etc.

En esta notación, después de **BIN** sólo podemos poner ceros y unos, de forma que el número sólo puede ser un entero no negativo. Por ejemplo, no se puede usar **BIN −11** para representar el −3 decimal; lo que sí podemos hacer es escribir **−BIN 11**. El número tampoco puede ser mayor que el decimal 65535; es decir, no puede constar de más de dieciséis bits. Si rellenamos un número binario con ceros por la izquierda, por ejemplo, **BIN 00000001**, **BIN** los ignora y trata el número como si fuera **BIN 1**.

Sección 33
Programas de ejemplo

Programas:

- Renumerador
- Reloj
- Rebotes
- Tele tenis

Renumeración de líneas de programa

Este programa es una ampliación de la opción **'Renumerar'** del menú de edición, pues permite elegir tanto el número inicial como el intervalo entre líneas. Mezcle (con **MERGE**) este programa con el que desee renumerar y después póngalo en marcha dando la orden **RUN 9000**.

En respuesta a las instrucciones **INPUT** de las líneas 9000 y 9010, introduzca el número que quiere que tenga la primera línea en el programa renumerado (margen de 1 a 9999) y el salto que deba haber entre los números de líneas consecutivas (margen de 1 a 9999).

```
9000 INPUT "Linea inicial",com
9010 INPUT "Intervalo",salto
9020 LET comalt= INT (com/256)
9030 LET saltoalt= INT (salto/2
     56)
9040 POKE 23413,com-256*comalt
9050 POKE 23414,comalt
9060 POKE 23415,salto-256*salto
     alt
9070 POKE 23416,saltoalt
9080 PRINT "Pulse EDIT y luego
     seleccione la opcion Renum
     erar"
```

Reloj

Este programa convierte el ordenador en un reloj analógico (y digital).

Para ponerlo en marcha dé la orden **RUN**. Luego introduzca la hora (de 1 a 12) y los minutos (de 0 a 59).

```
 10 DIM s(60): DIM c(60)
 20 BORDER 0: PAPER 0: BRIGHT 1
    : INK 7: CLS
 30 PRINT AT´10,1;"Espere un po
    co mientras calculo"
 40 PRINT AT 11,1;"unos senos y
     cosenos"
 50 GO SUB 370
 60 LET z$="00"
 70 CLS
 80 INPUT "Hora (0-12):  ";h
 90 INPUT "Minutos (0-59):   ";
    m
100 LET s=0: POKE 23672,0: POKE
     23673,0
110 IF h=12 THEN LET h=0
120 LET xc=112: LET yc=90: LET
    r=70: LET rh=r/2: LET rm=r*
    3/4: LET rs=r*5/6
130 CIRCLE xc,yc,r
140 INK 1
150 FOR i=0 TO 359 STEP 30
160 PLOT (r+1)*s(i/6+1)+xc,(r+1
    )* c(i/6+1)+yc
170 NEXT i
180 INK 4
190 OVER 1: GO SUB 500
200 GO SUB 470
210 GO SUB 440
220 LET tm= INT (( PEEK 23672+2
    56* PEEK 23673)/50)
230 IF s+1=tm THEN LET os=s: LE
    T s=s+1: GO TO 250
240 GO TO 220
```

```
250 IF s=60 THEN LET s=0: POKE
    23672,0: POKE 23673,0: LET
    om=m: LET m=m+1: GO TO 290
260 PLOT xc,yc: DRAW rs*s(os+1)
    ,rs*c(os+1)
270 GO SUB 440
280 GO TO 220
290 IF m=60 THEN LET m=0: LET o
    h=h: LET h=h+1: GO TO 330
300 PLOT xc,yc: DRAW rm*s(om+1)
    ,rm*c(om+1)
310 GO SUB 470
320 GO TO 260
330 IF h=12 THEN LET h=0
340 PLOT xc,yc: DRAW rh*s(oh*5+
    1),rh*c(oh*5+1)
350 GO SUB 500
360 GO TO 300
370 PRINT AT 14,0
380 FOR i=6 TO 360 STEP 6
390 PRINT ".";
400 LET s(i/6)= SIN ((i-6)* PI
    /180)
410 LET c(i/6)= COS ((i-6)* PI
    /180)
420 NEXT i
430 RETURN
440 PLOT xc,yc: DRAW rs*s(s+1),
    rs*c(s+1)
450 LET s$= STR$ (s): PRINT OVE
    R 0; AT 18,27; INK 4;":"; I
    NK 6;z$( TO 2- LEN (s$));s$
460 RETURN
470 PLOT xc,yc: DRAW rm*s(m+1),
    rm*c(m+1)
480 LET m$= STR$ (m): PRINT OVE
    R 0; AT 18,24; INK 2;":"; I
    NK 5;z$( TO 2- LEN (m$));m$
490 RETURN
500 PLOT xc,yc: DRAW rh*s(h*5+1
    ),rh*c(h*5+1)
```

```
510 LET ph=h: IF ph=0 THEN LET
    ph=12
520 LET h$= STR$ (ph): PRINT OV
    ER 0; INK 3; AT 18,22;" "(
    TO 2- LEN (h$));h$
530 RETURN
```

Rebotes

Este programa es un sencillo juego para una persona contra el ordenador.

Para ponerlo en marcha dé la orden **RUN**.

Opciones:

La tecla [←] mueve la raqueta hacia la izquierda.
La tecla [→] mueve la raqueta hacia la derecha.
La barra espaciadora entrega una 'vida' a cambio de una nueva pantalla.

Al transcribir el programa tenga en cuenta lo siguiente:

1. Las letras "**BBBBB...**" de las líneas 30 y 50 representan caracteres gráficos. Para obtenerlos se pulsa [GRAF] una vez, y luego se pulsa [B] las veces necesarias. Después se pulsa nuevamente [GRAF] para desactivar el modo gráfico.
2. Los números "**3333**" de la línea 210 también representan caracteres gráficos. Para obtenerlos se pulsa [GRAF] una vez, y luego se pulsa [3] cuatro veces. Después se pulsa nuevamente [GRAF].
3. La letra "**A**" de la línea 430 también representa un carácter gráfico. Para obtenerlo se pulsa [GRAF] y luego [A]. Después se pulsa nuevamente [GRAF].

```
10 BORDER 0: INK 0: PAPER 0: C
   LS : BRIGHT 1
20 GO SUB 560
30 LET b$="BBBBBBBBBBBBBBBBBBB
   BBBBBBBBB": REM 28 Bs
40 LET s$="
                    ": REM 32 espa
  cios
```

```
 50 PRINT AT 3,12; INK 7; FLASH
    1;"FRONTON"; FLASH 0; AT 6
    ,9; INK 1;"B"; INK 7;" = 20
    puntos"; AT 8,9; INK 4;"B"
    ; INK 7;" = 10 puntos"; AT
    10,9; INK 2;"B"; INK 7;" =
    5 puntos"
 60 PRINT AT 14,1; INK 4;"Pulse
    ESPACIO o DISP para ceder"
    ; AT 15,1;"una vida a cambi
    o de una"; AT 16,1;"pantall
    a nueva."
 70 PAUSE 200
 80 LET maximo=0
 90 LET tpuntos=0
100 LET vidas=5
110 LET puntos=0
120 CLS
130 INK 7: PLOT 12,13: DRAW 0,1
    60: DRAW 230,0: DRAW 0,-160
    : INK 0
140 PRINT AT 1,2; INK 1;b$; AT
    2,2; INK 4;b$
150 FOR r=5 TO 6: PRINT AT r,2;
    INK 2;b$: NEXT r
160 LET bx=9
170 PRINT AT 19,2; INK 6;"CUALQ
    UIER TECLA PARA EMPEZAR"; A
    T 17,4;"Mueva raqueta con <
    y >"
180 PAUSE 0
190 PRINT AT 19,2; INK 0;s$( TO
    28); AT 20,0;s$( TO 32); A
    T 17,4;s$( TO 24)
200 PRINT AT 21,0; INK 0;s$( TO
    32): GO SUB 540: GO TO 220
210 PRINT AT 20,bx; INK 0;" ";
    INK 5;"3333"; INK 0;" ": RE
    TURN
220 LET xa=1: LET ya=1: IF INT
    ( RND *2)=1 THEN LET xa=-xa
```

```
230 GO SUB 210
240 LET x=bx+4: LET y=11: LET x
    c=x: LET yc=y
250 REM bucle principal
260 IF puntos>1100 THEN GO TO 1
    10
270 IF INKEY$ =" " OR INKEY$ ="
    0" THEN IF vidas>1 THEN LET
     vidas=vidas-1: GO TO 110
280 LET xc=x+xa: LET yc=y+ya
290 REM explorar teclado
300 GO SUB 470
310 IF yc=20 THEN IF ATTR (yc,x
    c)=69 THEN PLAY "N1g": LET
    ya=-ya: LET yc=yc-2: IF xc=
    bx+1 OR xc=bx+4 THEN LET xa
    =-xa: LET xc=x+xa
320 IF yc=21 THEN PLAY "O3N7#d"
    : PRINT AT y,x;" ": GO TO 4
    50
330 GO SUB 470
340 IF yc=20 THEN GO TO 430
350 LET t= ATTR (yc,xc)
360 IF t=71 THEN GO TO 410
370 IF t=64 THEN GO TO 420
380 LET ya=-ya: LET xz=xc: LET
    yz=yc: LET yc=yc+ya: GO SUB
     510: IF t=66 THEN PLAY "N1
    e": LET puntos=puntos+5: LE
    T tpuntos=tpuntos+5: GO SUB
     540: GO TO 350
390 IF t=68 THEN PLAY "N1c": LE
    T puntos=puntos+10: LET tpu
    ntos=tpuntos+10: GO SUB 540
    : GO TO 350
400 IF t=65 THEN PLAY "N1a": LE
    T puntos=puntos+20: LET tpu
    ntos=tpuntos+20: GO SUB 540
    : GO TO 350
410 LET xa=-xa: LET xc=xc+2*xa:
     PLAY "N1f"
420 IF yc=1 THEN LET ya=1
```

```
430 PRINT AT y,x; INK 0;" "; AT
    yc,xc; INK 3;"A": LET x=xc
    : LET y=yc
440 GO TO 250
450 LET vidas=vidas-1: IF vidas
    =0 THEN GO TO 530
460 GO SUB 540: GO TO 220
470 LET a$= INKEY$
480 IF (a$= CHR$ (8) OR a$="6")
     AND bx>1 THEN LET bx=bx-1:
     GO SUB 210: RETURN
490 IF (a$= CHR$ (9) OR a$="7")
     AND bx<25 THEN LET bx=bx+1
    : GO SUB 210: RETURN
500 RETURN
510 IF yz=20 THEN RETURN
520 PRINT AT yz,xz; INK 0;" ":
    RETURN
530 GO SUB 540: PRINT AT 10,9;
    INK 7;"FIN DEL JUEGO"; AT 1
    2,4;"Puntos conseguidos: ";
    tpuntos: FOR i=1 TO 300: NE
    XT i: GO TO 90
540 IF tpuntos>maximo THEN LET
    maximo=tpuntos
550 PRINT AT 21,14; INK 6;"MAX.
     ";maximo; AT 21,1;"PUNTOS
    ";tpuntos; AT 21,24;"VIDAS
    ";vidas: RETURN
560 FOR i= USR "a" TO USR "b"+7
570 READ b
580 POKE i,b
590 NEXT i
600 RETURN
610 REM pelota
620 DATA 0,60,126,126,126,126,6
    0,0
630 REM ladrillo
640 DATA BIN 11111111
650 DATA BIN 10000001
660 DATA BIN 10111101
670 DATA BIN 10111101
```

```
680 DATA BIN 10111101
690 DATA BIN 10111101
700 DATA BIN 10000001
710 DATA BIN 11111111
```

Tele tenis

Este programa es una versión del más veterano de los juegos de ordenador. Para dos jugadores, o para uno contra el ordenador.

Para ponerlo en marcha dé la orden **RUN**. Luego pulse [1] o [2] para elegir el número de jugadores.

Opciones:

Jugador 1: [A] sube la raqueta; [Z] la baja.
Jugador 2: [K] sube la raqueta; [M] la baja.
Gana el primer jugador que consigue 15 puntos.

Al transcribir el programa tenga en cuenta lo siguiente:

1. Los números **"66"** de la línea 150 representan caracteres gráficos. Para obtenerlos se pulsa [GRAF] una vez, y luego se pulsa [6] dos veces. Después se pulsa nuevamente [GRAF] para desactivar el modo gráfico.
2. El número **"8"** de las líneas 150, 250 y 540 también representa un carácter gráfico. Para obtenerlo se pulsa [GRAF] y luego [8]. Después se pulsa nuevamente [GRAF].
3. La letra **"A"** de la línea 330 también representa un carácter gráfico. Para obtenerlo se pulsa [GRAF] y luego [A]. Después se pulsa nuevamente [GRAF].

```
10 PAPER 4: INK 0: BRIGHT 0: B
   ORDER 4
20 CLS
30 GO SUB 730
40 DIM x(2): DIM y(2): DIM p(2
   )
50 LET comp=1: LET sc1=0: LET
   sc2=0: LET z$="0"
60 PRINT AT 2,10; INK 7;"TELE
   TENIS"
70 PRINT AT 8,2;"]UNO O DOS JU
   GADORES (1/2)?"
80 LET i$= INKEY$
```

```
 90 IF i$="1" THEN PRINT AT 12,
    8;"A para subir"; AT 14,8;"
    Z para bajar": GO TO 120
100 IF i$ ="2" THEN PRINT AT 10
    ,3;"Jugador 1: A para subir
    "; AT 12,14;"Z para bajar";
     AT 14,3;"Jugador 2: K para
    subir"; AT 16,14;"M para b
    ajar": LET comp=0: GO TO 12
    0
110 GO TO 80
120 FOR i=0 TO 200: NEXT i
130 LET x(1)=2: LET y(1)=3
140 LET x(2)=29: LET y(2)=18
150 LET e$="8": LET f$="66"
160 PRINT AT 1,0;
170 GO SUB 400: REM borde super
    ior
180 FOR i=3 TO 19
190 PRINT AT i,0; INK 6;f$; INK
     0; TAB 30; INK 6;f$
200 NEXT i
210 PRINT AT 20,0;
220 GO SUB 400: REM borde infer
    ior
230 PRINT AT 0,0; INK 1;"Jugado
    r 1: 00"; AT 0,19; INK 1;"J
    ugador 2: 00"
240 LET n= INT ( RND *2)
250 FOR i=1 TO 2: PRINT AT y(i)
    ,x(i); INK i;"8"; AT y(i)+1
    ,x(i);"8": NEXT i
260 IF n=0 THEN LET xb=21: LET
    dx=1: GO TO 280
270 LET xb=19: LET dx=-1
280 LET yb=12: LET dy= INT ( RN
    D *3)-1
290 GO SUB 440: REM mover raque
    tas
300 LET oxb=xb: LET oyb=yb: LET
     scd=0
```

```
310 GO SUB 580: REM mover pelot
    a
320 PRINT AT oyb,oxb; INK 0;" "
330 PRINT AT yb,xb; INK 7;"A"
340 IF scd=0 THEN GO TO 290
350 PRINT AT yb,xb; INK 0;" "
360 GO SUB 380
370 GO TO 240
380 PRINT AT 0,11; INK 1;z$( TO
     2- LEN ( STR$ (sc1)));sc1;
     AT 0,30; INK 2;z$( TO 2- L
    EN ( STR$ (sc2)));sc2
390 RETURN
400 FOR i=1 TO 64
410 PRINT INK 5;e$;
420 NEXT i
430 RETURN
440 LET a$= INKEY$
450 IF a$="a" THEN LET p(1)=-1
460 IF a$="z" THEN LET p(1)=2
470 IF comp=1 THEN LET p(2)=(2*
    (y(2)<(yb))-(y(2)>(yb))): G
    O TO 500
480 IF a$="k" THEN LET p(2)=-1
490 IF a$="m" THEN LET p(2)=2
500 FOR i=1 TO 2
510 LET a= ATTR (y(i)+p(i),x(i)
    )
520 IF p(i)=2 THEN LET p(i)=1
530 IF a=32 THEN PRINT INK 0; A
    T y(i),x(i);" "; AT y(i)+1,
    x(i);" ": LET y(i)=y(i)+p(i
    )
540 PRINT AT y(i),x(i); INK i;
    "8"; AT y(i)+1,x(i);"8"
550 LET p(i)=0
560 NEXT i
570 RETURN
580 LET w= ATTR (yb+dy,xb+dx)
590 IF w=32 THEN LET xb=xb+dx:
    LET yb=yb+dy: RETURN
```

```
600 IF w=33 OR w=34 THEN LET dx
    =-dx: PLAY "V15O7N1g": LET
    dy= INT ( RND *3)-1: RETURN
610 IF w=38 THEN GO TO 640
620 IF w=37 THEN PLAY "V15O7N1c
    ": LET dy=-dy
630 RETURN
640 PLAY "O3V15#d": IF dx>0 THE
    N LET sc1=sc1+1: GO TO 660
650 LET sc2=sc2+1
660 LET d=(sc1=15)+2*(sc2=15):
    LET scd=1
670 IF d00 THEN GO SUB 380: PRI
    NT INK 7; AT 10,4;"El jugad
    or ";d;" gana."; AT 12,4;"]
    Jugar otra vez (s/n)?": GO
    TO 690
680 RETURN
690 IF INKEY$ ="" THEN GO TO 69
    0
700 IF INKEY$ ="s" THEN RUN
710 IF INKEY$ ="n" THEN STOP
720 GO TO 690
730 FOR i=0 TO 7
740 READ n
750 POKE USR "a"+i,n
760 NEXT i
770 RETURN
780 DATA 0,60,126,126,126,126,6
    0,0
```

Utilización de la calculadora

Temas tratados en este capítulo:

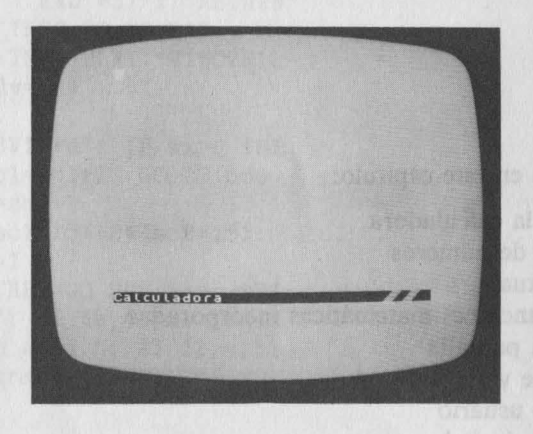
Selección de la calculadora
Introducción de números
Resultado actual
Uso de las funciones matemáticas incorporadas
Edición de la pantalla
Asignación de variables
Funciones de usuario
Salida de la calculadora

El **+2A** puede ser usado también como calculadora, con todas las funciones habituales en las calculadoras más potentes.

Para utilizar la calculadora, abra el menú de presentación y seleccione la opción **Calculadora**. (Si no sabe cómo seleccionar una opción de menú, consulte el Capítulo 2.)

La calculadora puede ser seleccionada en cuanto se enciende el **+2A**. O bien, si ya se está trabajando con +3 BASIC, se puede elegir la opción **Salida** del menú de edición para volver al menú de presentación y entonces seleccionar la opción **Calculadora**. El programa de BASIC con el que se estuviera trabajando al realizar esta selección será conservado en la memoria y restablecido cuando se abandone esta opción.

Cuando se selecciona la opción **Calculadora** la pantalla cambia a:



y la calculadora del **+2A** queda preparada para aceptar instrucciones. Escriba:

6+4

Tan pronto como pulse [INTRO], aparecerá la respuesta: **10**. (Observe que no se pulsa la tecla [=] como se haría en una calculadora corriente.)

Verá que el cursor está situado a la derecha de la respuesta, que es un *resultado actual* (como en una calculadora normal). Esto significa que podemos sencillamente escribir la siguiente operación para que sea aplicada al 'resultado actual' (sin tener que escribirlo de nuevo). Así, con el cursor todavía a la derecha del **10**, escriba:

/5

y obtendrá la respuesta: **2**. Ahora escriba:

***PI**

La calculadora responde con **6.2831853**. El **+2A** ha utilizado la función intrínseca π; todo lo que hemos tenido que hacer es escribir **PI**. Esto es válido para *todas* las funciones matemáticas del **+2A**. Para comprobarlo, escriba:

***ATN 60**

y obtendrá **9.7648943**. También es posible 'editar' el contenido de la pantalla. Desplace el cursor (con la tecla [←]) hasta el principio de la línea y escriba **INT**, de forma que la línea quede así:

INT 9.7648943

Cuando pulse INTRO, obtendrá la respuesta: **9**. Esto también demuestra que el **+2A** puede escribir el valor de una expresión aunque ello no implique la realización de ningún cálculo. Como ejemplo, pulse INTRO y después escriba:

1E6

y obtendrá el valor de esa expresión. Fíjese en que antes de escribir **1E6** ha sido necesario pulsar INTRO para indicarle al ordenador que íbamos a empezar un cálculo nuevo.

Una característica muy útil de la calculadora del **+2A** es que permite asignar valores a variables para usarlos en cálculos posteriores. Esto requiere el empleo de la sentencia **LET** (como en BASIC). Escriba lo siguiente:

LET x=10

(Es necesario pulsar INTRO *dos veces* para que el **+2A** acepte la asignación de la variable.) Ahora, para comprobar que el **+2A** reconoce la variable **x**, escriba lo siguiente:

x+90

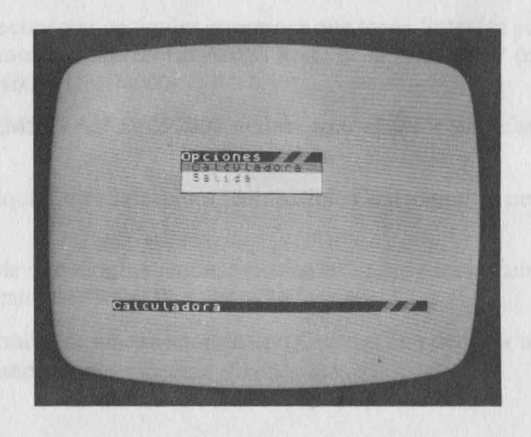
y a continuación:

+x*x

Si está usted usando la calculadora mientras tiene almacenado en la memoria un programa de BASIC, debe elegir las variables para la calculadora de forma que no entren en conflicto con las utilizadas por el programa.

No se puede usar las palabras clave de BASIC como nombres de variable.

Cuando termine de trabajar con la calculadora, pulse la tecla EDIT. La pantalla cambiará a:



Seleccione la opción **Salida** para volver al menú de presentación. Si estaba trabajando en un programa de +3 BASIC antes de empezar a usar la calculadora, puede volver al programa seleccionando la opción **+3 BASIC**. (Si desea seguir con la calculadora, seleccione la opción **Calculadora**.)

Si ha definido alguna función de usuario (con la sentencia **DEF FN**) mientras estaba en BASIC, puede invocarla desde la calculadora. Para comprobarlo vuelva a +3 BASIC y escriba, por ejemplo,

9000 DEF FN c(n)=n*n*n

Esta línea define una función que calcula el cubo de *n* (es decir, del número que pongamos entre los paréntesis). Ahora salga de +3 BASIC y vuelva a la calculadora. Para calcular el cubo de 3 basta con escribir:

FN c(3)

Periféricos para el +2A

Temas tratados en este capítulo:

Impresora
Joystick(s)
Monitor
Amplificador
Dispositivos serie
Dispositivo MIDI
Interfaz auxiliar
Dispositivos de ampliación
Unidades de disquete

El **+2A** puede ser conectado a una amplia gama de dispositivos adicionales (*periféricos*), tales como joysticks, impresora, unidades de disquete, etc. En este capítulo vamos a dar toda la información necesaria para conectarlos.

Impresora

Al **+2A** se le puede conectar cualquier impresora que tenga 'interfaz paralelo tipo Centronics'. Recomendamos las impresoras AMSTRAD de la serie DMP (modelos DMP2000, DMP3000, DMP3160 y DMP4000).

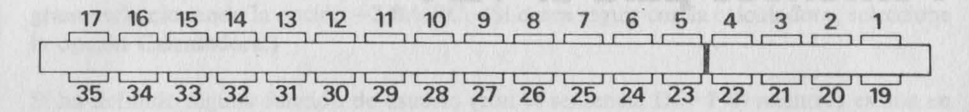
Para conectar la AMSTRAD DMP2000 se debe usar el cable suministrado con la propia impresora.

Para conectar cualquier otra impresora 'compatible Centronics' se necesita el cable AMSOFT PL-1.

El extremo del cable que termina en un conector estándar para circuito impreso debe ser insertado en el zócalo **IMPRESORA** del **+2A**.

El otro extremo termina en un conector de tipo Centronics y debe ser insertado en el zócalo de datos de la impresora.

Las conexiones del zócalo **IMPRESORA** son las indicadas en la figura y la tabla siguientes:



(visto por detrás)

Zócalo IMPRESORA

Patilla	Función	Patilla	Función
1	STROBE (negada)	19	masa
2	D0	20	masa
3	D1	21	masa
4	D2	22	masa
5	D3	23	masa
6	D4	24	masa
7	D5	25	masa
8	D6	26	masa
9	D7	27	no utilizada
10	no utilizada	28	masa
11	BUSY	29	no utilizada
12	no utilizada	30	no utilizada
13	no utilizada	31	no utilizada
14	masa	32	no utilizada
15	no utilizada	33	masa
16	masa	34	no utilizada
17	no utilizada	35	no utilizada
18	no existe	36	no existe

Aunque sólo hay 34 terminales, están numerados del 1 al 17 y del 19 al 35 (o sea, no existe el 18 ni el 36) para mantener la correspondencia con el zócalo de la impresora.

La impresora que se conecte al **+2A** debe generar por sí misma un avance de línea cada vez que reciba un retorno del carro. Generalmente esta función puede ser ajustada con uno de los 'conmutadores basculantes' de la impresora (en las AMSTRAD DMP, con el conmutador DS1-4).

Al **+2A** también se le puede conectar casi cualquier impresora de tipo 'serie' que se ajuste a la norma RS232. Recomendamos a los usuarios no expertos que se abstengan de experi-

mentar con las conexiones de la puerta serie. Para conectar una impresora serie se debe comprar un cable adecuado y seguir las instrucciones de instalación y funcionamiento del manual de la impresora. La conexión de la impresora serie se realiza en el zócalo **RS232/MIDI** del **+2A**.

Las operaciones con la impresora (tanto serie como paralelo) están descritas en las Secciones 21 y 22 del Capítulo 8.

Joysticks

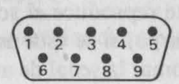
Con el **+2A** sólo se puede utilizar los joysticks Sinclair de la serie SJS. Cualquier otro tipo de joystick (por ejemplo, Atari) **no** operará directamente, ya que su clavija de conexión está cableada de forma diferente.

En el lateral izquierdo del **+2A** se encuentran los dos zócalos para joystick. En general, con los programas de juego sólo se usa el **JOYSTICK 1**.

Si un programa determinado le permite elegir el tipo de joystick, seleccione la opción 'Interface Two' (o 'Sinclair'), ya que los circuitos de joystick del **+2A** están diseñados para funcionar exactamente igual que el Interface Two.

Es importante no conectar ni desconectar el joystick cuando el **+2A** está encendido.

Patilla	Función
1	no utilizada
2	masa
3	no utilizada
4	disparo
5	arriba
6	derecha
7	izquierda
8	masa
9	abajo



Zócalos JOYSTICK1 y JOYSTICK2

Monitor

Además de un televisor (o en su lugar) el **+2A** puede utilizar un monitor monocromático o de color, o un televisor de sistema PERITEL. Si el monitor que usted desea usar no está anunciado como compatible con el Sinclair **+2A**, es muy probable que tenga que adquirir un cable especial para él; en tal caso, consulte con su distribuidor.

El monitor (o televisor PERITEL) debe ser conectado en el zócalo **RGB/PERITEL** del **+2A**.

Patilla	Función
1	+12 V
2	masa
3	salida de audio
4	sincr. compuesta
5	+12V
6	verde
7	rojo
8	azul



Zócalo RGB/PERITEL

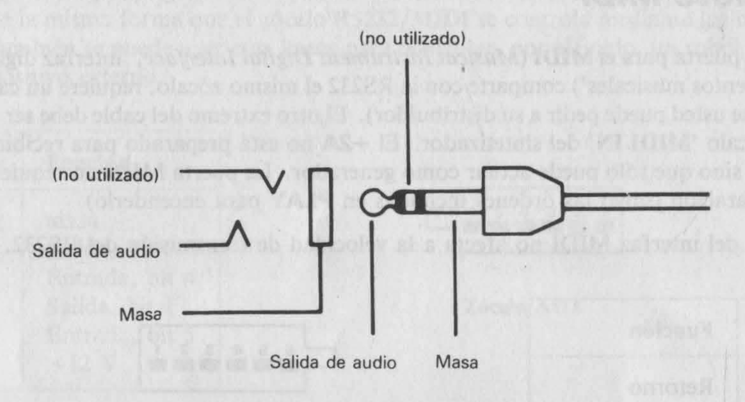
Al utilizar un monitor se ha de prever qué se desea hacer con el sonido. Si el monitor tiene entrada de audio, se la puede conectar a la patilla 3 del zócalo **RGB/PERITEL** o al zócalo **CINTA/SONIDO** del **+2A**; si el monitor no admite sonido, se necesitará un amplificador externo (véase el apartado siguiente).

Amplificador

El **+2A** normalmente reproduce el sonido a través del aparato de televisión al que está conectado. Sin embargo, si se está usando un monitor o si se desea grabar o amplificar el sonido, se puede tomar la señal de audio del zócalo **CINTA/SONIDO** de la parte posterior del **+2A**. (Es una clavija monoaural de 3.5 mm; la señal es de 200 mV pico a pico sobre impedancia de 5 kΩ.) Cuando se utiliza un amplificador, conviene tener en cuenta que las señales de 'carga' y 'grabación' del magnetófono también son enviadas a la salida **CINTA/SONIDO**; por consiguiente, se debe bajar el mando de volumen del amplificador al mínimo mientras se efectúa estas operaciones.

Otro detalle que se debe observar es que el nivel de sonido producido por **BEEP** es equivalente al de los tres canales de **PLAY** funcionando al mismo tiempo. En la práctica, esto significa que **BEEP** sonará bastante más alto que **PLAY**, lo cual puede causar problemas si los niveles de sonido son críticos.

No se debe conectar ni desconectar nada en el zócalo **CINTA/SONIDO** mientras el **+2A** esté encendido.



Zócalo CINTA/SONIDO

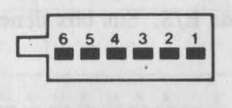
Las órdenes de control del sonido están descritas en la Sección 19 del Capítulo 8.

Dispositivos serie

Para conectar cualquier otro dispositivo serie al **+2A** se necesita un cable de tipo 'serie para Spectrum **+2A**'; pídaselo así a su distribuidor.

Por si desea preparar usted mismo el cable, las conexiones son las siguientes:

Patilla	Función
1	masa
2	TXD
3	RXD
4	DTR
5	CTS
6	+12 V



Zócalo RS232

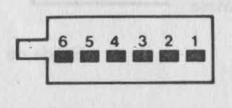
El control de los dispositivos serie está descrito en la Sección 21 del Capítulo 8.

Dispositivo MIDI

Aunque la puerta para el **MIDI** (*Musical Instrument Digital Interface*, 'interfaz digital para instrumentos musicales') comparte con la RS232 el mismo zócalo, requiere un cable diferente (que usted puede pedir a su distribuidor). El otro extremo del cable debe ser conectado al zócalo '**MIDI IN**' del sintetizador. El **+2A** no está preparado para recibir datos del MIDI, sino que sólo puede actuar como generador. La puerta MIDI no requiere ninguna preparación (salvo las órdenes incluidas en **PLAY** para encenderlo).

El empleo del interfaz MIDI no afecta a la velocidad de transmisión del RS232.

Patilla	Función
1	Retorno
2	no utilizada
3	no utilizada
4	no utilizada
5	Salida de datos
6	no utilizada



Zócalo MIDI

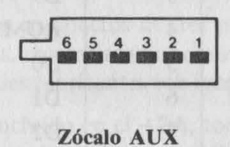
Interfaz auxiliar

El zócalo **AUX** (interfaz auxiliar) dispone de dos líneas de entrada (patillas 3 y 5) y de dos líneas de salida (patillas 2 y 4). Las líneas de E/S son controladas por los circuitos 1488 y 1489, que a su vez están conectados a las líneas de E/S del AY-3-8912 (véase la hoja de especificación de este dispositivo). Básicamente, el registro 16 del AY-3-8912 controla las ocho líneas de E/S. Sus bits tienen el siguiente significado:

Bit	Función
0	**AUX**, patilla 2 (salida)
1	**AUX**, patilla 4 (salida)
2	**RS232**, patilla 5 (CTS, salida)
3	**RS232**, patilla 3 (RXD, salida)
4	**AUX**, patilla 3 (entrada)
5	**AUX**, patilla 5 (entrada)
6	**RS232**, patilla 4 (DTR, entrada)
7	**RS232**, patilla 5 (TXD, entrada)

Por programa se puede gestionar las líneas de E/S como si fuesen una segunda puerta RS232 (de la misma forma que el zócalo **RS232/MIDI** se controla mediante los bits 2, 3, 6 y 7). También se puede usar esas líneas para controlar, por ejemplo, un robot o algún otro dispositivo externo.

Patilla	Función
1	masa
2	Salida, bit 0
3	Entrada, bit 4
4	Salida, bit 1
5	Entrada, bit 5
6	+12 V

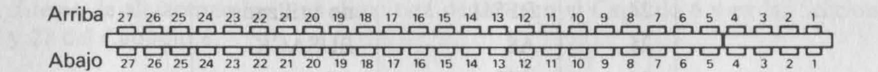


Zócalo AUX

Dispositivos de ampliación

El **+2A** puede ser conectado a una gama muy amplia de periféricos a través del zócalo **E/S EXPANSION** de la parte posterior del aparato. Aunque este zócalo es muy parecido al del antiguo Spectrum de 48K, no se puede garantizar que un dispositivo que funcione correctamente con éste lo haga igual con el **+2A**. Así pues, antes de comprar cualquier dispositivo de ampliación, debe usted cerciorarse de que funciona con el **+2A**.

> ¡ATENCION! Es muy peligroso conectar o desconectar cualquier dispositivo en el zócalo **E/S EXPANSION** estando el **+2A** encendido. Si lo hace, es probable que estropee tanto el ordenador como el dispositivo externo.



Zócalo E/S EXPANSION

Patilla	Fila superior	Fila inferior
1	A15	A14
2	A13	A12
3	D7	+5V
4	ROM 1 $\overline{OE}$	no utilizada
5	D0	masa
6	D1	masa
7	D2	CK
8	D6	A0
9	D5	A1
10	D3	A2
11	D4	A3
12	$\overline{INT}$	no utilizada
13	$\overline{NMI}$	masa
14	$\overline{HALT}$	ROM 2 $\overline{OE}$
15	$\overline{MREQ}$	DISK $\overline{RD}$
16	$\overline{IORQ}$	DISK $\overline{WR}$
17	$\overline{RD}$	MOTOR ON
18	$\overline{WR}$	$\overline{BUSQR}$
19	no utilizada	RESET
20	$\overline{WAIT}$	A7
21	+12V	A6
22	−12V	A5
23	$\overline{M1}$	A4
24	$\overline{RFSH}$	no utilizada
25	A8	BUSACK
26	A10	A9
27	RESET	A11

Para mayor información sobre el hardware del **+2A** se puede consultar la Sección 30 del Capítulo 8.

Unidades de disquete

Si desea conectar una o dos unidades de disquete al **+2A**, debe adquirir el modelo AMSTRAD FD-1, junto con el interfaz adecuado (AMSTRAD SI-1). El SI-1 se conecta en el zócalo **E/S EXPANSION** de la parte trasera del **+2A** y puede controlar hasta dos unidades FD-1.

La unidad AMSTRAD FD-1 sólo admite 'discos compactos de tres pulgadas'. Para garantizar la integridad de los datos y los programas, recomendamos el uso de discos CF-2 de AMSOFT; no obstante, los discos de los principales fabricantes son igualmente adecuados.

Gracias a la versatilidad del sistema operativo incluido en el **+2A**, todas las operaciones de mantenimiento de ficheros (copia, borrado, etc.) pueden ser realizadas con una sola unidad de disquete. Sin embargo, la segunda unidad facilita y acelera estos procesos y reduce notablemente el riesgo de accidentes.

Antes de conectar o desconectar las unidades de disquete, se debe extraer los discos y asegurarse de que el equipo está apagado.

> ¡ATENCION! Es muy peligroso conectar o desconectar el interfaz de la unidad de disco en el zócalo **E/S EXPANSION** estando el **+2A** encendido. Si lo hace, es probable que estropee tanto el ordenador como el interfaz.

Cuando se tiene conectada una FD-1, se debe encender **primero** la FD-1 (con el interruptor deslizante que está en su panel posterior) y **después** el **+2A** (enchufando el cable de la fuente de alimentación). Se iluminará el piloto de alimentación (verde) de la FD-1.

Si además está conectada la segunda unidad externa, se debe encender ambas unidades antes que el **+2A**. Se iluminarán los pilotos de alimentación (verdes) de las dos unidades, y además el de lectura/escritura (rojo) de la segunda unidad.

El sistema de almacenamiento en disco está descrito en el Capítulo 6 y en las Secciones 20 y 27 del Capítulo 8.

Unidades de disquete

Si desea conectar una o dos unidades de disquete al +2A, debe adquirir el modelo AMSTRAD FD-1, junto con el interfaz adecuado (AMSTRAD SI-1). El SI-1 se conecta en el zócalo E/S EXPANSION de la parte trasera del +2A y puede controlar hasta dos unidades FD-1.

La unidad AMSTRAD FD-1 sólo admite discos compactos de tres pulgadas. Para garantizar la integridad de los datos y los programas, recomendamos el uso de discos CF-2 de AMSOFT; no obstante, los discos de los principales fabricantes son igualmente adecuados.

Gracias a la versatilidad del sistema operativo incluido en el +2A, todas las operaciones de mantenimiento de ficheros (copia, borrado, etc.) pueden ser realizadas con una sola unidad de disquete. Sin embargo, la segunda unidad facilita y acelera estos procesos y reduce notablemente el riesgo de accidentes.

Antes de conectar o desconectar las unidades de disquete, se debe extraer los discos y asegurarse de que el equipo está apagado.

> ¡ATENCION! Es muy peligroso conectar o desconectar el interfaz de la unidad de disco en el zócalo E/S EXPANSION estando el +2A encendido. Si lo hace, es probable que estropee tanto el ordenador como el interfaz.

Cuando se tiene conectada una FD-1, se debe encender primero la FD-1 (con el interruptor deslizante que está en su panel posterior) y después el +2A (enchufando el cable de la fuente de alimentación). Se iluminará el piloto de alimentación (verde) de la FD-1.

Si además está conectada la segunda unidad externa, se debe encender ambas unidades antes que el +2A. Se iluminarán los pilotos de alimentación (verdes) de las dos unidades, y además el de lectura/escritura (rojo) de la segunda unidad.

El sistema de almacenamiento en disco está descrito en el Capítulo 6 y en las Secciones 20 a 27 del Capítulo 8.

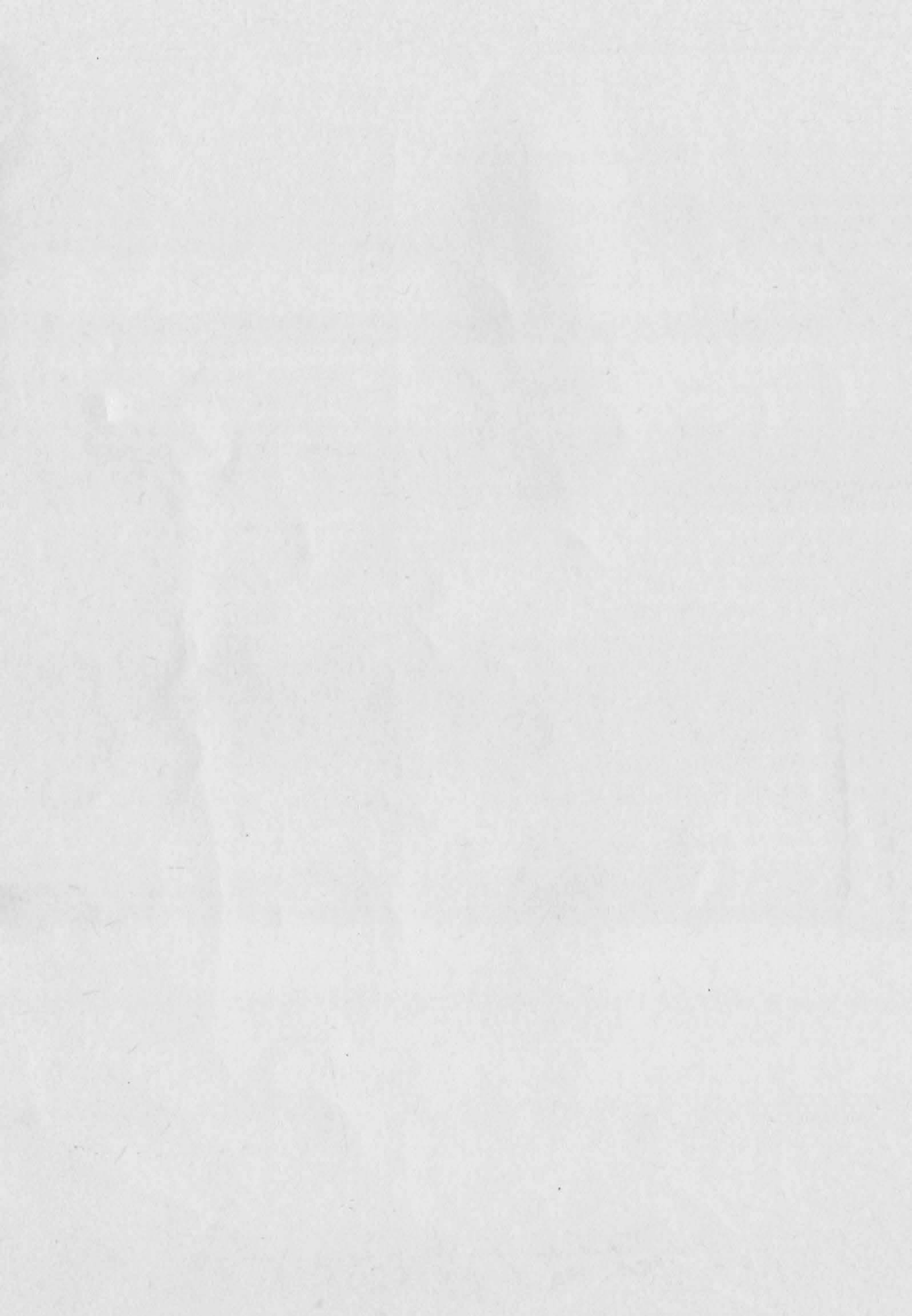
Índice

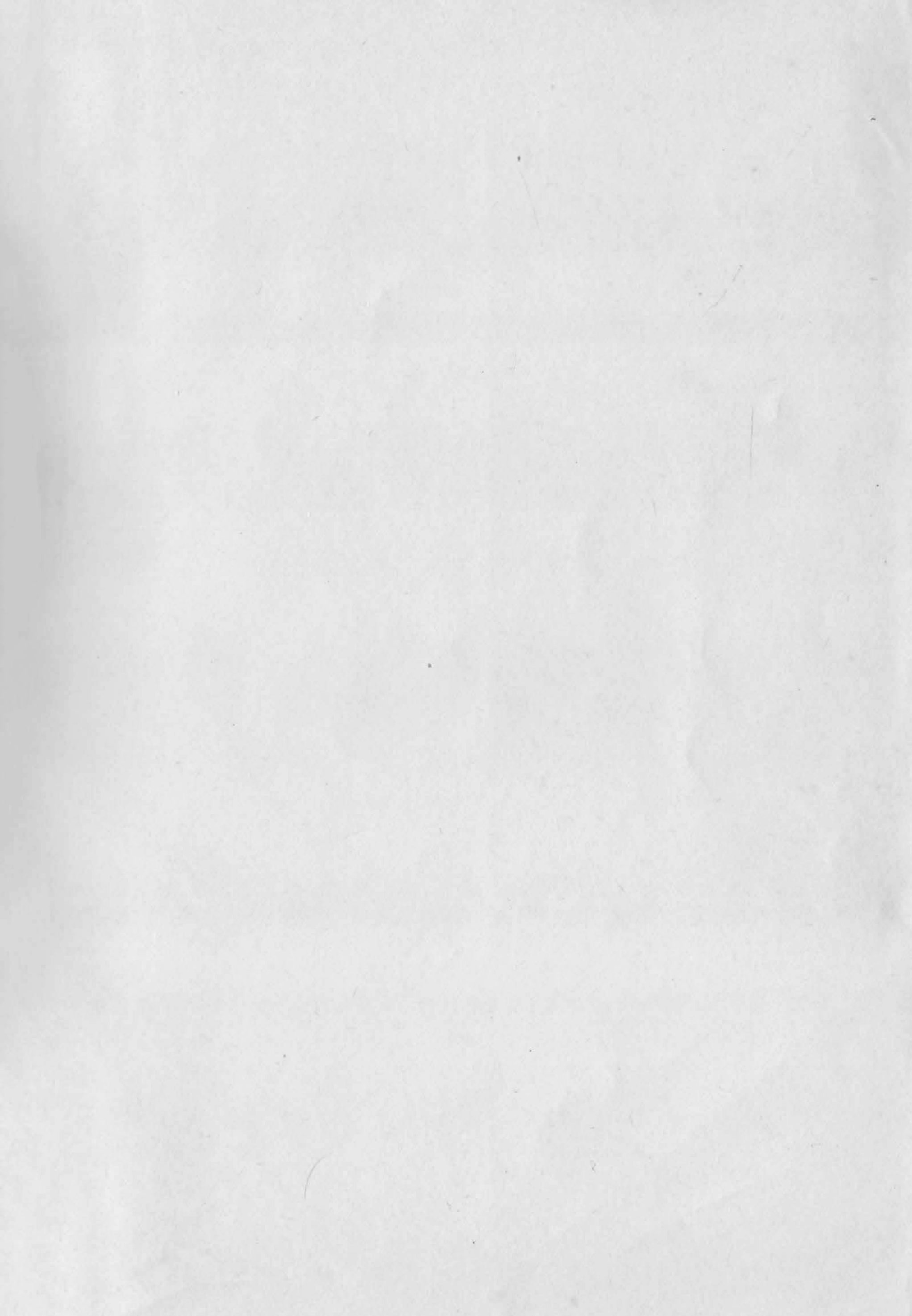
P

R

S











AMSTRAD ESPAÑA

ARAVACA, 22, 28040 MADRID

TELEFONO 459 30 01

TELEX 47660 INSC E

FAX 459 22 92